



PHD

High-level interpretive languages and real-time programming.

Thompson, J. W.

Award date:
1977

Awarding institution:
University of Bath

[Link to publication](#)

Alternative formats

If you require this document in an alternative format, please contact:
openaccess@bath.ac.uk

Copyright of this thesis rests with the author. Access is subject to the above licence, if given. If no licence is specified above, original content in this thesis is licensed under the terms of the Creative Commons Attribution-NonCommercial 4.0 International (CC BY-NC-ND 4.0) Licence (<https://creativecommons.org/licenses/by-nc-nd/4.0/>). Any third-party copyright material present remains the property of its respective owner(s) and is licensed under its existing terms.

Take down policy

If you consider content within Bath's Research Portal to be in breach of UK law, please contact: openaccess@bath.ac.uk with the details. Your claim will be investigated and, where appropriate, the item will be removed from public view as soon as possible.

HIGH-LEVEL INTERPRETIVE LANGUAGES
AND REAL-TIME PROGRAMMING

Submitted by J.W. Thompson

for the degree of Ph.D.

at the University of Bath

1977

COPYRIGHT

"Attention is drawn to the fact that copyright of this thesis rests with its author. This copy of the thesis has been supplied on condition that anyone who consults it is understood to recognise that its copyright rests with its author and that no quotation from the thesis and no information derived from it may be published without the prior written consent of the author".

"This thesis may be made available for consultation within the University Library and may be photocopied or lent to other libraries for the purpose of consultation".

J.W. Thompson

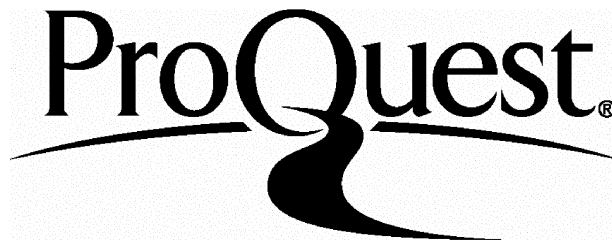
ProQuest Number: U442953

All rights reserved

INFORMATION TO ALL USERS

The quality of this reproduction is dependent upon the quality of the copy submitted.

In the unlikely event that the author did not send a complete manuscript and there are missing pages, these will be noted. Also, if material had to be removed, a note will indicate the deletion.



ProQuest U442953

Published by ProQuest LLC(2015). Copyright of the Dissertation is held by the Author.

All rights reserved.

This work is protected against unauthorized copying under Title 17, United States Code.
Microform Edition © ProQuest LLC.

ProQuest LLC
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106-1346

ACKNOWLEDGMENTS

I would like to thank the following people and organisations for their contributions to the preparation of this thesis.

My supervisors, Paul Sawyer and Alan Charlesworth, for their valuable advice and constructive comments throughout my period of research and during the compilation of the thesis.

Tony Lockett, for his support in the hardware field, without him the project would have been impossible.

The Professor and members of staff of the Department of Chemical Engineering at the University of Bath for providing the opportunity and facilities for my research and study.

The Science Research Council for providing financial support.

The typist, Gillian Lockett, who had the difficult task of translating a hand-written manuscript which I could understand into something which other people could understand.

SUMMARY

This thesis describes the design and development of high level language programming systems for use on a small mini-computer installation in an educational environment.

Any high level language may be implemented in a number of different ways. The major alternatives are reviewed with particular reference to hardware resources, ease of use and the need to cater for both on-line and off-line applications.

The implementation of a programming system based upon the use of an interpreter rather than a compiler is described. The system enables a single user to create develop and execute programs for applications involving on-line data acquisition and control.

Hardware resources are most effectively utilized in a time shared rather than a dedicated mode. The problems of extending this system to permit time sharing are discussed and an appropriate solution is described.

In a situation where it is necessary to segregate data processing from data acquisition, the use of a common programming system is desirable. An off-line operating system is described which adds file handling capabilities to the interpreter and provides a common software interface between data acquisition and data processing programs.

CONTENTS

i.

	Page
CHAPTER 1. A Survey of Programming Systems	1
1.1. Introduction	2
1.2. On Line Programming Systems	4
1.2.1. Block Diagrammatic Programming Systems	4
1.2.2. Fill in the Form Programming Systems	7
1.2.3. Compiler Based High Level Language Systems	7
1.2.4. Interpreter Based High Level Language Systems	11
1.3. Operating Systems	12
1.4. Conclusions	12
 CHAPTER 2. Preliminary Design Considerations	 13
Choice of System Software	14
2.1. Hardware Configuration	14
2.2. Applications and Uses	14
2.3. Data Acquisition Rates	15
2.4. Review of Existing Software Systems	16
2.4.1. Fill in the Form Programming	16
2.4.2. Compiler Based Operating Systems	17
2.4.3. Interpreter Based Operating Systems	18
2.5. Conclusions	19
2.6. Multi-Access Interpretive Programming	19
2.7. Extensions of Interpreter Facilities for Off-Line Purposes	 20
 CHAPTER 3. The FOCAL Interpreter	 22
3.1. Introduction	23
3.2. Rules and Syntactical Limitations	23
3.3. Text Storage and Handling Routines	26
3.4. Variable Search and Storage Routines	27

	Page
3.5. Interpretation	29
3.6. Expression and Argument Evaluation	29
3.7. Pushdown List Controls	31
3.8. Execution of a Program	31
3.9. Error Diagnostics	33
3.10. Editing Facilities	33
3.11. Control Pointers for FOCAL Programs	33
3.12. Speed of Operation of a FOCAL Program	33
3.13. The Interrupt Processor	35
3.14. Character Input and Output	35
 CHAPTER 4. Initial Modifications to FOCAL	 37
4.1. Reconfiguration of FOCAL Software	38
4.2. Saving and Editing Variable Tables	39
4.2.1. Editing Variable Tables	41
4.2.2. Saving Variable Tables	41
4.3. Input Buffer Overflow	42
4.4. Enhancements to the MODIFY Command	43
4.5. Hard Copy Facilities for the ASK Command	45
4.6. Protection Systems	48
4.6.1. Command Buffer Overflow	48
4.6.2. Power failure Protection	48
4.7. New Functions	49
4.8. Summary	49
 CHAPTER 5. A Single User Real-Time Programming System	 50
5.1. Real-Time Programming	51
5.2. The PDP-8 Computer and Interface System	52
5.2.1. Device Selection and Channel Addressing	53
5.2.2. Input Devices	54
5.2.3. Output Devices	55

	Page
5.2.4. Other Peripheral Devices	56
5.2.5. Interrupt System	56
5.2.6. System Clock	56
5.2.7. Fault Protection System	58
5.3. Real-Time Software	58
5.3.1. Current or Elapsed Time	58
5.3.2. Synchronous Data Scanning	61
5.3.3. Program Priority System	63
5.3.3.1. FOCAL Break Points	64
5.3.4. Input/Output Transfers	66
5.3.4.1. Clock Driven Peripherals	66
5.3.4.2. Directly Accessed Peripherals	67
5.3.5. Control Functions	71
5.3.5.1. Lead-Lag Function for Control	71
5.3.5.2. PCI Control Algorithm	76
5.3.5.3. Timing and Core Allocation of the Control Algorithm	79
5.3.6. Communication with Operating Program	80
5.3.7. Fault Protection	81
5.3.8. Error Detection	81
5.3.9. Timing of Functions	83
5.4. Conclusions	83
 CHAPTER 6. A Time Shared Real-Time FOCAL System	 91
6.1. Time Sharing the FOCAL Interpreter	92
6.2. Design Constraints	93
6.3. User Exchange Techniques	93
6.4. Terminal Input/Output Handling	94
6.5. Core Allocations and Design Criteria	95
6.6. Executive Routines of the Time Shared System	99
6.6.1. The Interrupt Processor	99

	Page
6.6.2. User Status Record Manipulation	101
6.6.3. Keyboard Service Routine	102
6.6.4. Printer Service Routine	104
6.6.5. Character Printing Routine ACTION Q	105
6.6.6. User Swapping Routine	105
6.6.6.1. Character Input Routine to the Interpreter XRD	105
6.6.6.2. Character Output Routine from the Interpreter EXPRN	105
6.6.6.3. User Swapping Points Check Routine EXCHK	105
6.6.6.4. New User Selection Routine EXCHE	106
6.7. Summary	106
6.8. Modifications to the Interpreter to allow for proposed core modifications	109
6.8.1. LIBRARY Command	109
6.8.2. Symbol table Editing and Saving	109
6.8.3. MODIFY Command	109
6.8.4. Hard copy for ASK	110
6.8.5. Random Number Generator FRAN	110
6.8.6. Other Differences	110
6.9. Timing Considerations	110
6.10. Priority Tasking System	112
6.11. Input/Output Transfer	112
6.12. Control	113
6.13. Parameter Modification Routine	114
6.14. Error Code	115
6.15. Initial Dialogue	115
6.16. Conclusions	116
 CHAPTER 7. A File Structured FOCAL Operating System	 122
7.1. Introduction	123
7.2. Structure of a File Monitoring System	124
7.3. Requirements of the Monitor System	125

v.

	Page
7.4. DECTape Programming	127
7.5.1. DECTape Controlling Routine for the Monitor system	128
7.5.2. Use of the Controlling Routine	133
7.6. File Directory System	135
7.6.1. Naming of Files	135
7.6.2. File Data stored in Directory	136
7.6.3. Length of Directory	137
7.6.4. Additional Directory Information	137
7.6.5. Directory Handling Routine	138
7.7. Interrupt Processor	139
7.8. Structure of the Command Decoder	142
7.8.1. Command Structure	142
7.9. System Commands	145
7.9.1. DIRECTORY	145
7.9.2. SAVE	146
7.9.3. LOAD	148
7.9.4. ERASE	149
7.9.5. RUN	150
7.9.6. COPY	151
7.9.7. FIND	151
7.9.8. ZERO	151
7.10. Special Monitor Commands	152
7.10.1. HELP	152
7.10.2. Interprocessor Communication	153
7.10.2.1. WRITE and ACCESS	154
7.11. Bootstrapping the System	155
7.12. Program Chaining Facilities provided by the system	156
7.13. Discussion	158

	Page
CHAPTER 8. Conclusions and Recommendations	175
8.1. Introduction	176
8.2. Choice of Operating System	177
8.3. Real-Time FOCAL Systems	178
8.3.1. Hardware Limitations	178
8.3.2. Software Limitations	180
8.4. FOCAL as an Off-Line Programming Language	182
8.5. Use of Interpretive Programming Systems	183
8.6. Conclusions	183
<u>REFERENCES</u>	185
APPENDIX A. Use of Real-Time FOCAL	191
A.1. Introduction	192
A.2. Equipment and Experimental Procedure	192
A.3. Choice of Objective Functions	193
A.4. Lattice Search Procedure	194
A.5. Results	194
A.6. On-Line Search Procedures	195
A.6.1. Simplex Search Procedure	196
A.6.1.1. Results	197
A.6.2. Random Search Procedure	198
A.6.2.1. Results	198
A.6.3. Conclusions	198
APPENDIX B. Effects of Errors in Timing upon the design consideration of a Real-Time Operating System	236
B.1. Methods of Providing Synchronous Scanning Facilities	237
B.2. Types of Timing errors	238
B.2.1. Effects produced by missing a complete sampling period	239

	Page
B.2.2. Effects produced by random errors in Sampling times "Jitter Sampling"	239
B.3. Simulation Studies	241
B.3.1. Data Generation	241
B.3.2. Least Squares Analysis of Data	242
B.3.3. Spectral Analysis of Data	243
B.4. Results and Conclusions	245
APPENDIX C. Flow Charts for FOCAL Modification	266
APPENDIX D. Flow Charts for Single User Real-Time FOCAL	276
APPENDIX E. Flow Charts for Two User Real-Time FOCAL	298
APPENDIX F. Flow Charts for the File Monitor System	329

CHAPTER 1.

A Survey of Programming Systems

1.1. Introduction

One of the basic requirements of a computer system, whether operating on-line for data acquisition and control purposes or off-line for data processing, is that users should be able to operate the system efficiently with a minimum amount of effort and experience. This is especially true in an education and research environment where applications are generally of a one-off nature and potential users have only a limited amount of time available for program development.

For off-line data processing, high level languages such as FORTRAN, ALGOL, BASIC and FOCAL* have become well established. These languages allow users to express mathematical operations as a series of instructions in a form which is similar to their natural manner of expression.

High level languages can be implemented by the use of either a compiler based system or an interpreter based system.

In the case of a compiler based system, high level source code statements are converted into machine object code by means of a translating program or compiler. The compiled program can then be loaded into the computer and executed as required. The processes of translation and execution are therefore two distinct steps which must proceed sequentially.

Interpretive systems are based upon the use of a core resident interpreter program and high-level language statements which are stored as text within computer memory. The interpreter program scans through the text interpreting and executing it statement by statement. Each statement in the users program is essentially a subroutine call to the section of the interpreter providing the function required by the statement. Statements are linked together by the executive part of the interpreter in the manner demanded by the statement sequence within the text (1). In this case, therefore, the translation and execution of the program proceed in parallel. Figure 1. illustrates the differences between various modes of computer programming.

In contrast to the high level language systems which have been implemented for off-line data processing, systems for on-line data

*FOCAL is a registered trade mark of Digital Equipment Corporation.

Machine CodeAssembly CodeHigh Level Language
Compiler Based
(Fortran)

001010100000

TAD D

111000100001

CIA

001010100001

TAD B

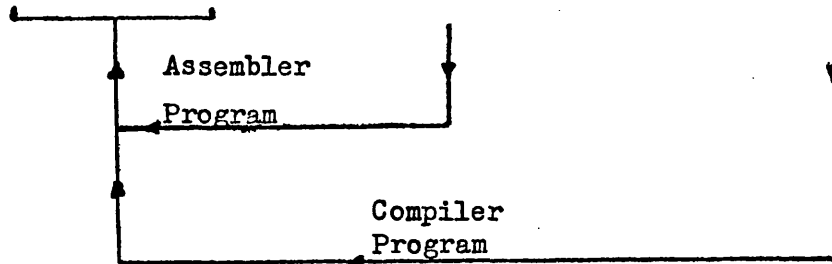
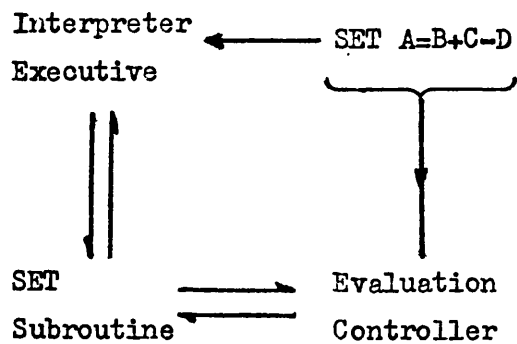
A=B+C-D

001010100010

TAD C

011010100011

DCA A

High Level Language
Interpreter Based
(Focal)Figure 1 : A Comparison of Programming Languages

acquisition, processing and control have in the past been implemented by programming at a low level in assembler or machine code (2). This has been mainly due to the need for obtaining efficiency in the use of memory and processing time which have been difficult to accomplish when using high level languages. Programming at a low level does, however, have the disadvantages of being both tedious and time consuming. In addition, considerable experience is required before effective applications programs can be developed by a user.

With the current trends in computer hardware technology, the costs of computers and memory in particular are falling rapidly (3), whilst the cost of expertise in the programming field is rising by comparison. It has therefore become both feasible and desirable to use a high level language system for on-line applications whenever possible.

The use of a high level language system for on-line applications suffers from two main disadvantages. Firstly, program execution speeds are limited particularly in the case of an interpreter based system. Secondly, it is difficult to provide users with the capabilities for truly synchronous timing of events (4).

These disadvantages would appear, however, to be outweighed by the advantage gained from the ease of use provided by such a system.

1.2. On-line Programming Systems

The following approaches have evolved in order to make on-line programming easier from the users point of view.

1. Block Diagrammatic Programming (5 - 9)
2. Fill in the Form Programming (10 - 13)
3. Compiler based high level languages (14 - 23)
4. Interpreter based high level languages (1,24 - 28)

1.2.1. Block Diagrammatic Programming Systems

This concept of programming is based upon the use of a set of software "building blocks" analogous to those used by an engineer in the design of a control system. A control program is constructed from

these blocks by specifying, as data, block parameter and inter block linkages. The data is entered into the computer system from a special purpose hardware communication device.

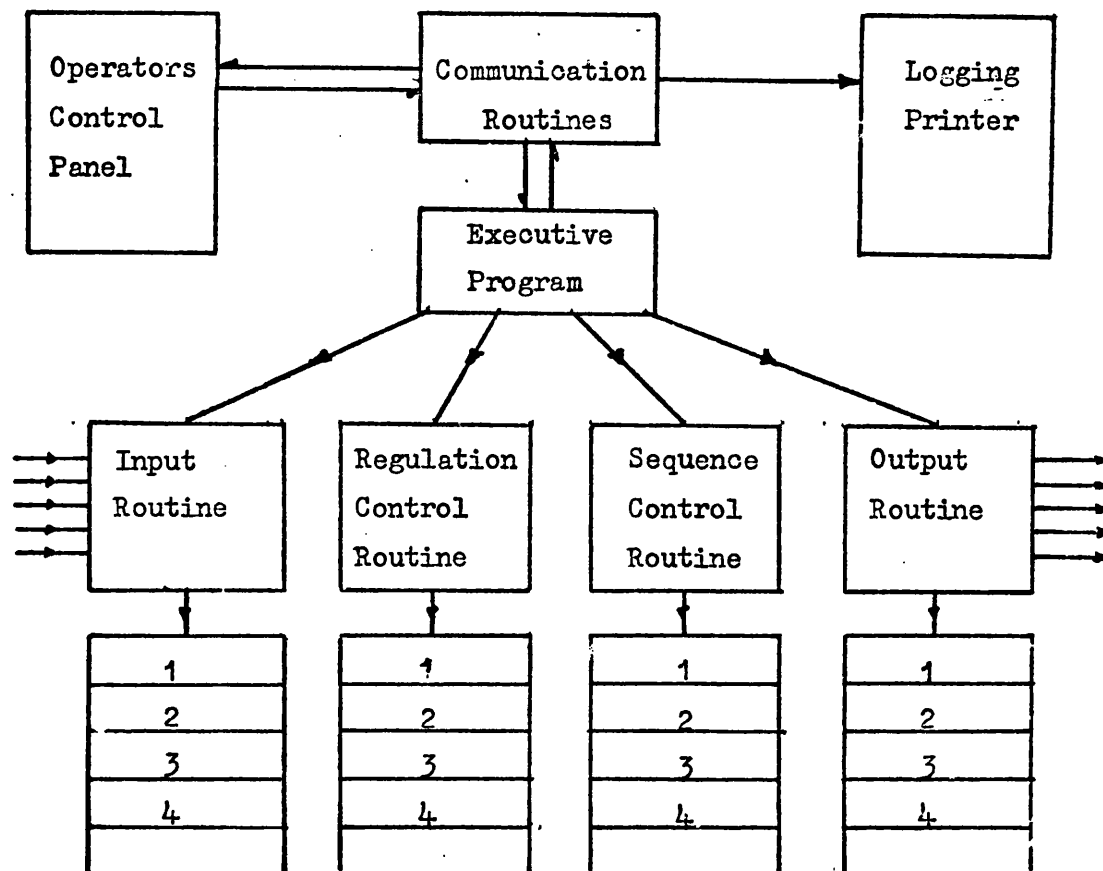
The structure of CONRAD III (8), a typical example of this form of programming, is illustrated in Figure 2. together with typical core requirements for such a system.

The programming skill necessary to use such a system is minimal because it is an extension of an established design procedure and because user/system communication is simplified.

There are however a number of disadvantages associated with this approach. Amongst these are:-

- (a) The need to provide a special purpose hardware communication device and associated software.
- (b) The use of fixed block parameter tables results in inefficient memory utilization.
- (c) A large number of blocks are required to implement complex control strategies, as blocks have to be cascaded together to provide the necessary functions.
- (d) The block diagrammatic approach is inappropriate for applications involving data acquisition and processing in addition to control.

As a result block diagrammatic programming would appear to be most suited for direct digital control applications rather than for general on-line programming. The substantial memory requirements of such a system (8) also precludes their use on small minicomputer systems.



Input
Data

Regulation
Control
Block
Parameters

Sequence
Control
Block
Parameters

Output
Data

Storage Requirements

Communication 6K, Executive 6K, Subroutines 2K

Data Tables Require Core in Addition to above values

Figure 2 : Schematic Diagram of CONRAD III Operation

1.2.2. Fill in the Form Programming Systems

This approach to on-line programming is also based upon the concept of a set of basic system subroutines. As its name implies, however, parameters and linkages for the blocks are specified by the user on pre-printed forms. The information from these forms is then transferred to punched cards or tapes for input to the system.

Typical data input forms used in I.B.M.'s PROSPERO (10) are shown in Figures 3, 4, and 5.

Fill in the Form programming, like Block Diagrammatic programming, is more applicable to the direct digital control of industrial processes than to general on-line programming.

1.2.3. Compiler based High Level Language Systems

Systems of this type fall into two main categories (29) :-

- (a) Procedural Programming Languages, such as FORTRAN and ALGOL which have been enhanced so that a user may program on-line tasks such as timing, data acquisition and control in addition to data processing. The original syntax of the language is retained in the enhancements.
- (b) Problem-orientated languages such as AUTRAN (29), whose syntax has been designed specifically for the purpose of programming the control of items of plant equipment. The commands and statements of such a language is therefore closely related to items of plant.

For example, AUTRAN uses statements of the form

START (PUMPA), CONFIRM

which when compiled and executed, instructs the computer to start the pump motor given the identification PUMPA and to confirm that the motor has started.

Alarm conditions may be set up by using statements of the form

WHEN (PUMPA. ALARMS. OFF) SCHEDULE (SHUTDOWN)

which instructs the computer to execute a shut down procedure should PUMPA fail to operate in the correct manner.

IBM 1800 PROSPRO III

Variable Information for Supervisory Control

1	9
C	

Description

[illegible]

* Entries marked with an asterisk in most cases must be filled in for DDC variables.

** Bumpless transfer options: 1 = only supervisory, 2 = supervisory and DDC, 3 = hold old target.

GX10-0025-0 U/M 025

Figure 3

IBM 1800 PROSPRO III

General Action, Supervisory

Description

7 8	10 11	13	18	22	40	Calling Source and Remarks	80
0 0	G A		1				

13: Are values beyond MAX-MIN limits acceptable? (0=No, 1=Yes)

18-22: Next General Action to be executed. (Chaining)

Step	Code		A		B			Remarks	80
7 8	10	12 14	16	20	22	26	40		
0 1	S				-				
0 2	S				-				
0 3	S				-				
0 4	S				-				
0 5	S				-				
0 6	S				-				
0 7	S				-				
0 8	S				-				
0 9	S				-				
1 0	S				-				
1 1	S				-				
1 2	S				-				
1 3	S				-				
1 4	S				-				
1 5	S				-				
1 6	S				-				

RTN - Return to processing
 END - End Variable Processing
 CON - Constant
 MSG - Type Message No./Value
 PVS - Process Variable Special
 FBA - Feedback Adjustment
 FFA - Feedforward Adjustment
 VSQ - Variable Status Query
 SRT - Save Real Time
 CDT - Calculate Delta Time
 VEE - Variable Equals Equation
 VEV - Variable Equals Variable
 VEC - Variable Equals Constant
 VTV - Variable Times Variable
 XSP - Execute Special Program
 RCS - Read Contact Point
 XCO - Operate Contact Point

VTC - Variable Times Constant
 VDV - Variable Divided by Variable
 VPV - Variable Plus Variable
 VPC - Variable Plus Constant
 VMV - Variable Minus Variable
 ABS - Absolute Result
 CVV - Compare Variable to Variable
 CVC - Compare Variable to Constant
 BRM - Branch Result Minus
 BRZ - Branch Result Zero
 BRP - Branch Result Plus
 BCL - Branch Compare Low
 BCE - Branch Compare Equal
 BCH - Branch Compare High
 BRA - Branch Unconditionally
 GET - DDC Item
 PUT - DDC Item
 PCS - Put Supervisory Cascade Status

GX10-0027-0 U/M 025

Figure 4

IBM 1800 PROSPRO III

Variable Information for DDC

1	5
0	

Description

7 8	10		DDC processing phase
6 0			Routine DDC process interval (1, 2, 4, 8, 16, 32, 64 sec)
6 1			Measured input? (1=Yes)
6 2			Measured input processing sub-routine (0=none)
6 3			General Block executed for input calculation
6 4	4		Input filter type (0=none, 1=Union, 2=exp., 3=slope)
6 5			Input filter factor (with decimal point)
6 6			Time delayed input? (1=Yes)
6 7			
7 0	4		General Block executed after input is processed
7 1			Setpoint value (sign and dec. pt. in eng. units)
7 2			Deviation for control action (eng. units w/dec. pt.)
7 3			Input change vs. output change: inc.-inc. (0), inc.-dec. (1)
7 4			Output (0=stored-only, 1=valve, 2=cascade, 3=link)
7 5			DDC station or variable no. for cascade or link
7 6			Output maximum limit { Positive values to nearest 5%,
7 7			Output minimum limit { Negative values to nearest 10%, up to 100%
7 8			Output change limit (1=1%, 2=5%, 3=10%, 4=100%)
7 9			Reload in last output state or manual? (1=manual)
8 0			Controller setpoint no. for setpoint tracking
8 1			Type of loop (0=none, 1=standard)
8 5			Loop gain constant (sign and decimal point)
8 6			Integral or reset constant in min. (w/dec. pt.)
8 7			Derivative or lead constant in min. (w/dec. pt.)
8 8			Nonlinear gain constant (with decimal point)
8 9			Omit proportional mode? (1=Yes)
9 0			Ramp to new setpoint? (1=Yes)
9 1			Nonlinear proportional mode? (1=Yes)
9 2			Nonlinear reset mode? (1=Yes)
9 5	4		General Block executed after output is stored
9 6			First Associated Data Block number
9 7			Last Associated Data Block number

Figure 5

The operation of a compiler based system is essentially a multi-stage process involving:-

- (a) Generation of the program source code
- (b) Translation of the source code into machine code instructions
- (c) The execution of the machine code program

The serial nature of these processes entails that only the required segment of the system is core resident at a particular time. The memory overhead required by system software is therefore significantly lower than that required by a system which is totally core resident. The major disadvantage of the compiler based system arises when users programs require modification during the development stage. This involves repeating each of the separate processes of the compiler system until a successful program execution has been achieved.

The efficiency of most compiler based systems is such that program execution speeds and memory requirements are comparable with those of programs written at assembler level.

1.2.4. Interpreter based High Level Language Systems

As in the case of compiler based systems, interpreter based systems are either extensions of existing languages (BASIC, FOCAL) or special purposes languages (e.g. NODAL).

The implementation of a language by an interpreter system results in a slow speed of program execution by comparison with that achieved when using a compiler. This is due to the fact that a program statement must be re-interpreted each time it is executed (1). It is also necessary to retain the entire interpreter system in memory during both program development and execution. This results in a reduction of space available for users programs when dealing with a fixed size minicomputer system.

On the other hand, the resident nature of the interpreter together with the mode of text storage has the advantage that users can be provided with a range of aids to program development and testing. These

include powerful interactive text editing facilities and the ability to execute individual statements directly in an "immediate mode". Users program development time can therefore be significantly reduced in comparison to the development time required when using a compiler based system.

1.3. Operating Systems

Irrespective of the type of programming system adopted, some form of Real-Time operating system or executive must be employed to control the various input/output operations of the computer system, to share effectively the resources available between competing programs and users on a priority basis and to control individual program execution. In the case of a dedicated computer system, this can be achieved by a simple interrupt processor and run-time system. When however the resources of a computer system have to be time shared between a number of users, the complexity of the executive (1,8,17,30,33) demands that hardware priority interrupt systems and direct memory access mass storage must be available (1).

1.4. Conclusions

It can be seen therefore that the software structure appropriate to on-line programming will be determined by the following factors:-

- i. The facilities provided by the central processor
- ii. The type and quantity of main memory and mass storage available
- iii. The range of input/output peripherals available
- iv. The maximum data acquisition rates required of the system
- v. The type of users and applications for which the system must cater.

CHAPTER 2.

Preliminary Design Consideration

2.. Choice of System Software

The factors influencing the choice of system software have been identified in the previous section. Before deciding upon which type of software structure should be adopted, it was necessary to examine the various factors with respect to the particular situation.

2.1. Hardware Configuration

A PDP-8 minicomputer (34) with 8K of core memory was available. Although the associated Kent K70 (35,36) interface provided a large range of input/output peripheral devices for signal processing, the absence of any mass storage device and the primitive interrupt system available within the processor imposed severe constraints upon the design of an operating system.

The core only configuration of the computer meant that the development of a multi-access system would be virtually impossible. It would therefore be advantageous if program development time could be minimized so as to produce a rapid turnover of system users. This could be achieved by including powerful program editing and debugging facilities within the system executive.

2.2. Applications and Users.

It was envisaged that any computer system developed would be used for a variety of purposes both on-line and off-line. In particular for on-line applications, the system was expected to be used for the data acquisition from and control of experimental equipment varying from a single instrument to pilot plant. In addition to this, it was expected that facilities for on-line data processing would be an essential feature.

For effective teaching of topics such as engineering design it was considered that an interactive off-line computer system would be necessary.

These considerations meant that users would be expected to generate most of their own applications programs. As users would either be students or members of staff from the department, the time they could devote to creating working programs would be limited. In

order to reduce program development time therefore, an important feature of any computer operating system would be to design the system primarily for ease of use. The use of a common language for both on and off-line applications would also help to reduce programming time.

2.3. Data Acquisition Rates

It is difficult to accurately assess in advance likely data acquisition rates required of a laboratory computer system, although certain guide lines may be used to obtain estimates of possible sampling rates.

For control purposes the common variables which are required are flow, temperature pressure and composition. Typical sampling rates for these variables are as follows (37):-

FLOW	0.1 seconds to 1.5 seconds
PRESSURE	1.0 seconds to 5 seconds
TEMPERATURE	10 seconds to 20 seconds
COMPOSITION	> 20 seconds

Another possible application for the system is that of the analysis of chromatographic data. The scan rates required to recover the area under a single peak is dependent upon the elution time of the component in question. However it would appear that about 10 samples per peak are required to effect better than 0.25% accuracy in peak area recovery (38). Since for most chromatographic applications minimum peak widths are of the order of 5 to 10 seconds, a sample rate of about two points per second would be required.

Baumann et al (39) suggests that a scan rate of five points per second would be sufficient to provide satisfactory area recovery for most chromatographic applications.

Signal analysis problems or the analysis of data obtained from instruments like a mass spectrometer generally require high data capture rates. However, these tend to be specialised problems and hence require specialised software to handle the high data rates rather than the

more generalised approach required of a laboratory computer system. It would however be advantageous if the computer operating system could be modified as necessary in order to handle bursts of data such as would be expected from signal analysis problems. The operating system could then be used for system initialisation purposes before linking to a machine code subroutine used to obtain the data. Data processing could then take place subsequently by returning to the operating system.

It would appear therefore that for most laboratory applications a maximum data rate of about 10 to 20 samples per second would be sufficient. This magnitude of sample rate would not exclude any of the available high level language operating systems.

2.4. Review of Existing Software Systems

The possibility of designing an entirely new operating system was considered. This however would have involved the implementation of ideas, procedures and data processing methods used in existing systems. It was therefore decided that the most effective solution to the problem would be to develop an operating system using as a basis, the framework already provided within an existing operating system. Any major deficiencies in such a system could be remedied to make the system applicable to off-line use for the situation in question. Enhancements would also be required in order to provide facilities for on-line use.

2.4.1. Fill in the Form Programming

A fill in the form programming system, PROCON (36), was available for use on the PDP-8. However, experience gained in using this system and also in extending PROCON for sequence control (40) emphasised the inflexibility of the Fill in the Form approach. For example, a study of the sequence control of a simple boiler system (40) showed that any user would have to become proficient at assembler level programming before being able to use such a system. The limited error diagnostic and correction facilities available within the system made the process of program development difficult.

The most severe disadvantage of this system was the absence of any data processing facilities, the system having been designed

specifically for direct digital control purposes.

It was therefore decided that this form of operating system could not provide the necessary facilities and flexibility required even if large scale modifications of the system were undertaken.

2.4.2 Compiler Based Operating Systems

For an 8K PDP-8 core only computer FORTRAN II (41) was available. This is in effect a multi pass system operating in the following manner:-

1. A punched paper tape in ASCII format is created using the text creation program SYMBOLIC EDITOR (41).
2. The paper tape source code is then translated into a symbolic language SABR (11) using the FORTRAN compiler program. This produces another punched paper tape in ASCII format.
3. The symbolic punched paper tape is translated into a relocatable binary coded punched paper tape by using the SABR assembler program.
4. The relocatable binary tape is loaded into memory together with any required library routines by using the LINKING LOADER.

At this stage the program is ready for execution. The process has however involved the loading of four binary coded paper tape programs. SYMBOLIC EDITOR, the FORTRAN compiler, the SABR assembler program and the LINKING LOADER. In addition, each separate stage involves the generation of one or more punched paper tapes to be reloaded in the subsequent stage. Library routines are loaded in the final stage of the process. A minimum total of ten paper tapes must therefore be loaded into the system.

Errors in syntax and logic have to be corrected by returning to the first stage of the process in order to edit the source code program with the SYMBOLIC EDITOR.

The above version of FORTRAN is a typical example of the implementation of compiler based systems on core only minicomputers.

The use of such a system implies that a user must become fully conversant with the operating procedures of the computer and would therefore violate the basic design requirement of being easy to use.

Adapting such a system for the provision of on-line facilities also presents a large problem. The segmented nature of the system would necessitate that modifications be made to most of the sections. The compiler must be enhanced to decode new commands from source code into SABR code. New library routines would have to be developed in SABR code for loading into the system prior to running the program. The LINKING LOADER runtime system would also have to be extended to provide the additional interrupt service routines required for on-line use.

2.4.3. Interpreter Based Operating Systems

The 8K FOCAL/69 (47,48) interpreter was available for the PDP-8 as an off-line programming system. The main advantage of this system is the way in which it operates. Once the system has been loaded, users can create and edit programs from a keyboard terminal and then execute the programs without the need for loading separate segments of the operating system. Paper tapes are only required for the initial loading of the system and for use as a storage medium for users programs.

The "load and go" mode of operation of the interpreter, although simplifying the process of program development and execution, does require that the whole of the interpreter remains core resident at all times. The space available for extending such a system to provide on-line capabilities in a core only configuration is therefore limited. An acceptable compromise between space available for text storage and space required for system enhancements would therefore have to be achieved if such an interpreter were to be used as an on-line operating system.

Another advantage of the FOCAL interpreter is that it possesses a command set which is similar in nature to that used in FORTRAN, a language which is most commonly used for off-line data processing. The transition between FORTRAN and FOCAL would therefore present few problems to users.

2.5. Conclusions

Consideration of the above factors indicated that an interpreter system would provide a suitable basis for an on-line operating system under the following constraints:-

- (a) The system is to be developed primarily for ease of use
- (b) Sampling rates required are in general not high
- (c) No mass storage memory device is available

This conclusion has also been arrived at by a number of other researchers in the position of trying to provide on-line computing facilities in a laboratory environment (1,28,42,43,44,45). For larger computer installations it would appear that the compiler based systems take precedence (33,46), NODAL (24) being a notable exception.

A preliminary investigation of the FOCAL interpreter indicated that the system could be modified and with suitable re-arrangements would allow sufficient core space for most of the modifications required. The main problem with such a system would be one of achieving speeds of program execution compatible with the sampling rates required.

2.6. Multi-Access Interpretive Programming

As outlined earlier in the discussion, the limited capacity and primitive interrupt structure of the PDP-8 virtually eliminated the possibility of creating an efficient multi-access high level language operating system.

A preliminary investigation of the FOCAL interpreter indicated that such facilities could perhaps be provided to a limited extent by including a user swapping executive within the interpreter system. This would effectively allow the interpreter to scan and execute different users programs on a roll-in/roll-out basis. Severe restrictions would however be imposed upon the size of the programs which could be accommodated as the available core would have to be shared by system users.

The worst feature of such a system would be the reduced execution speeds incurred by sharing available processing time between system users. The interpretive mode of execution was expected to be slow for a single

user system and the performance of the system would be reduced even further by attempting to share processing time.

However, as most applications would appear to require relatively slow sampling rates, it was thought to be possible to time share FOCAL, thus enabling the advantages of the interpretive system to be retained.

Before attempting to develop a multi-user interpretive system, it would first be necessary to assess the performance of a single user system. Should it prove to be a satisfactory operating system, it could be extended and re-arranged to provide similar facilities for more than one user.

2.7 Extensions of the Interpreter Facilities for Off-Line Purposes

The major drawbacks of an interpretive system for general computational purposes in a core only configuration system are:-

- i. The limited size of program which can be accommodated.
- ii. The use of a paper tape as a storage medium both for system software and for user developed software.

With a certain amount of re-arrangement of the FOCAL interpreter it could be made possible for users to chain various program segments together. This would provide a modular program structure which can be an advantage in situations where it is advisable to examine the sensibility of generated data before attempting further analysis. Intermediate data and program segments would however have to be saved on paper tape, adding to the problem of software storage in the core only system.

This problem could be overcome by the use of some form of magnetic mass storage device, allowing all software to be saved and reloaded simply and efficiently.

The purchase of a 12K PDP-8/E computer so as to extend the computing facilities within the department meant that a processor was available for off-line computation. The choice of a mass storage device for the system was influenced by the need to provide more than a single drive device for software security and hence backup in the case of failure. The cost of providing either a duplicate disc drive or disc drive and magnetic tape was beyond available finance. Thus a dual drive

TU56 DEC tape system and TD8E DEC tape controller (50) was purchased as being the only system capable of providing the necessary backup and security within the agreed budget.

The system unfortunately has the disadvantage of not being a direct memory access device. All data transfers must therefore take place through the accumulator of the PDP-8/E under program control, precluding its use during real-time operations. Its use was therefore limited to one of an external storage device for system and user created software thereby avoiding the need to use paper tape as a storage medium.

A powerful off-line operating system, OS8, was available for such a computer configuration. This provides excellent high level language programming capabilities in the form of FORTRAN and BASIC and extensive file handling facilities for program storage and use.

Unfortunately the provision of such extensive facilities inevitably means that the system is complex and its efficient operation requires that a user has knowledge and experience of computer operation.

In an educational environment where the computer is used as a tool and users are expected to develop their own programs, it is essential that programming is made as simple as possible. It was decided that the use of OS8 would require too much effort on behalf of the user and that a much simpler system could be developed using FOCAL as a basic programming language. The decision was also influenced by the advantages which would be gained by using a common language for both on and off-line programming.

CHAPTER 3.

The FOCAL Interpreter

3.1. Introduction

A description of the FOCAL language and a brief guide to the operation of its internal subroutines was available (47). Unfortunately there was insufficient detail available in the manual to enable modifications to the system to be made efficiently. It was therefore necessary to examine the PAL III assembler listing of 4K FOCAL/69 in order to determine the structure and method of operation of the interpreter. The flow charts (51) produced from the listing are included on microfiche.

Figures 6 and 7 illustrate the core utilization of the 4K FOCAL and the 8K FOCAL systems respectively.

3.2. Rules and Syntactical Limitations

FOCAL programs are created by entering command strings into the computer from the keyboard of a terminal device. The rules which have to be obeyed are as follows:-

1. All indirect program lines must be numbered according to group number and line within the group, XX.YY where

$1 \leq XX \leq 31$ is the group number

$01 \leq YY \leq 99$ is the line number within the group

Hence line numbers are in the range 01.01 to 31.99 excluding XX.00 .

2. A line may contain any number of commands (except WRITE, MODIFY and ERASE) separated by semi colons. The line is completed with a carriage return.

A command has the form

(COMMAND) (SPACE) (CHARACTER STRING) (TERMINATOR)

The command name only requires the correct first character to specify the command. The command name must be followed by a space, to act as a delimiting character between the command name and the character string.

4K FOCAL CORE MAP

Zero Page Pointers									
0	Command Deccdcr				Getln				
1	Do				Pushdown List Controls				
2	Write				Sortc				
3	Control & Transfer				Grptst				
4	If				Dys & Int				
5	Type & Ask				Adc,Outl and others				
6	Getarg				Spnor,Testn,Ran,Popj and others				
7	Ecall								
10	Terms Sgn,Abs et al				Delete				
11	Erase				Getc				
12	I33,Prntln,Prnt and Printo				Packe				
13	Interrupt Processor , I/O Routines and Error Recovery Routine								
14	Rubout				O/P buffer Command Buffer				
15									
16									
17									
20					TEXT VARIABLES AND PUSHDOWN LIST				
21									
22									
23	Exp				Log				
24					Atn				
25	Cos and Sin								
26					FLOATING POINT				
27					INPUT AND OUTPUT				
30					ROUTINES				
31					High Speed Reader Routine				
32									
33					FLOATING POINT INTERPRETER				
34									
35									
36	Sqt				Library				
37					Loaders				

Figure 6

0	8K FOCAL CORE MAP Field 0										
1	Zero Page Pointers										
2	Command Decoder					Getln					
3	Entabf		Do			Pushdown List Controls					
4	Control & Transfer		Write			Tests		Sorte		Grptst	Input
5	Comlst		If		Set & For			Dys & Int		Congo	
6	Type & Ask				Modify			Sortj		Adc, Outl and others	
7	Getarg							Spncr, Testn, Ren, Popj and others			
8	Ecall										
9	Terms		Egn, Abs et al		Tstlpr, Partest		Delete			Reade	Entabl
10	Erase		Findln			Getc			Endln		
11	I33, Prmtln, Prnt and Printc					Pcke					
12	Interrupt Processor, I/O Routines and Error Recovery Routine										
13	Rubout			Symbol Table Dump				C/P buffer			
14											
15											
16											
17											
18	VARIABLES AND PUSHDOWN LIST										
19											
20											
21											
22											
23			Exp						Atn		
24							Log				
25			Cos and Sin								
26	FLOATING POINT										
27	INPUT AND OUTPUT										
28	ROUTINES										
29											
30											
31						High Speed Reader Routine					
32											
33	FLOATING POINT INTERPRETER										
34											
35											
36	Sgt					Library					
37	Loaders										

	8K FOCAL CORE MAP Field 1									
0	Command Input Buffer									
1										
2										
3										
4										
5										
6										
7										
10										
11										
12										
13										
14										
15										
16										
17										
20	TEXT									
21										
22										
23										
24										
25										
26										
27										
30										
31										
32										
33										
34										
35										
36										
37	Loaders									

Figure 7

The syntactical form of the character string is determined by the particular command being executed, non-conformation to the particular restrictions imposed by the command structure causes an error condition. Each command must be terminated either with a semi colon or with a carriage return.

3. Variables are specified by a one or two character name, the first one of which must be an alphabetic character other than F, and an integer subscript in the range ± 2047 . All variables are floating point variables.
4. All functions, i.e. algorithms for producing trigonometric functions etc., begin with the letter F. The arguments required by the function are enclosed within parentheses immediately following the function name.
For example the function call FCOS (WT) evaluates the cosine of the variable identified by the symbol WT.
5. Any command which does not have a line number in front of it is executed immediately.

3.3. Text Storage and Handling Routines

As commands can be either entered into the system as a series of indirect commands preceded by statement numbers or as direct commands without a statement number, it is necessary to have an intermediate buffer between character input from a terminal keyboard and program storage within the text buffer. The intermediate buffer is known as the command input buffer.

Input characters from the console are stored within the command input buffer in stripped packed ASCII format (47). This form of storage generally reduces the 8 bit ASCII character code to a 6 bit code so as to economise in storage requirement. When a carriage return character is detected on input, the initial characters of the input string are examined

in order to determine whether the command is a direct or indirect command. If no line number is found, the command is executed immediately. Alternatively, if a line number is found, the four figure number is reformed into a 12 bit code; the most significant five bits for the group number between 1 and 31 and the least significant seven bits for the line number between 1 and 99, e.g. :-

08.76 would give the following 12 bit binary code

DECIMAL		BINARY	COMBINED CODE
Group 08	=	01000)	010,001,001,100
Line 76	=	1001100)	2 1 1 4 8

This 12 bit code, together with the rest of the character string held in the command input buffer, is added on to the current end of the text buffer. Any previous line entered into the text buffer with the same line number as the new line is immediately deleted and the space occupied by that line is recovered.

The new line is linked into its correct numerical order using line manipulation routines. A pointer at the start of the next lower numbered line is set to the starting address of the new line in the text buffer. Another pointer at the start of the new line is set to the starting address of the next higher numbered line. A further pointer is then incremented so as to maintain a record of the next free space available within the text buffer. Fig.8 shows a simple example of the method of text storage which allows numbered lines to be typed in in any sequence.

3.4. Variable Search and Storage Routines

Variables encountered during the execution of a FOCAL program are searched for in the area of core allocated to the variable storage, using a re-entrant variable search routine (GETARG). If the variable is found within the existing table, its new value is entered into the table. If the variable is not found within the table, the variable name, subscript and floating point value are appended to the end of the variable table. A pointer is used to maintain a record of the next free location in the variable storage area.

Program

01.01 S A=1

02.02 S B=2

03.03 S C=3

16.90 T A,B,C,

Text entered in the order of

01.01, 03.03, 02.02, 16.90

Program Storage in Text Buffer

<u>Location</u>	<u>Contents</u>	
0100	0114	Pointer to start of next sequential line 02.02
0101	0201	12 bit code for line number 01.01
0102	2340	S
0103	0175	A=
0104	6177	1 carriage return character occupies two
0105	1500	six bit bytes (code 7715)
0106	0122	Pointer to start of next sequential line 16.90
0107	0603	12 bit code for line number 03.03
0110	2340	S
0111	0375	C=
0112	6377	3
0113	1500	
0114	0106	Pointer to start of next sequential line 03.03
0115	0402	12 bit code for line number 02.02
0116	2340	S
0117	0275	B=
0120	6277	2
0121	1500	
0122	0000	Zero signifies highest numbered line in buffer
0123	4132	12 bit code for line number 16.90
0124	2440	T
0125	0154	A,
0126	0254	B,
0127	0354	C,
0130	4177	
0131	1500	
0132	First free location in buffer will be 0132	

Figure 8 : An Example of FOCAL Text Storage

Each variable occupies five full words comprised of one word containing two stripped ASCII characters for the variable name; one word containing a single subscript value in the range -2047 and a three word floating point number (one for the binary exponent and two for the mantissa). Fig.9.

3.5. Interpretation

The interpretation of the source code within FOCAL is essentially "table driven", i.e. there is a table in core which contains a list of valid command characters and a second table of dispatch addresses.

To interpret individual commands, FOCAL picks up the stripped character from the source code and reforms them into full 8 bit ASCII code. The first non-space character is tested against the list of valid command characters (COMLST). If a match is found at the m^{th} location in the list, the address in the m^{th} location in the second list (COMGO) is used as the address to which control must be transferred. A non matching character will cause an error condition.

In addition to the lists for commands, there are also lists for functions (FNTABL, FNTABF), terminators (TERMS) and various other operations within FOCAL. This type of table driven system makes it particularly easy to make modification and add extensions to the FOCAL language.

3.6. Expression and Argument Evaluation

All arithmetic operations within FOCAL are done in three (or four) word floating point binary arithmetic. These operations are controlled by a re-entrant evaluation routine. This routine, ECALL, ensures that higher priority operations are completed before lower priority operations and uses a software push down list in order to hold the lower priority tasks in abeyance.

Function calls are initiated from this routine by using hash coded function names, Fig.10, and "sort and branch" tables. On entering the subroutine associated with the particular function required, the first argument of the function call will have been evaluated and its value set

<u>Location</u>	<u>Contents</u>	
4000	0101	Variable name AA in stripped packed ASCII
4001	0000	Subscript value 0
4002	0001	
4003	2000	Three word floating point value 1.0
4004	0000	
4005	0261	Variable name B1 in stripped packed ASCII
4006	0010	Subscript value 8
4007	7775	
4010	3146	Three word floating point value 0.1
4011	3146	
4012	3200	Variable name Z in stripped packed ASCII
4013	0100	Subscript value 64
4014	0004	
4015	2400	Three word floating point value 10.0
4016	0000	

Figure 9 : An Example of FOCAL Variable Storage

Function call FINK hash coding = 2613

<u>Character</u>	<u>ASCII code</u>	<u>Hash code generation</u>
I	0311	
N	0316	$((0311 * 2) + 0316) * 2 + 0313$
K	0313	$=2613$

Figure 10 : Generation of Function Call Hash Code

up in the floating point accumulator. Evaluation of further arguments in the function call can be achieved by recalling ECALL to determine the value of a particular expression. On completion of the function algorithm, the floating point accumulator is generally loaded with the computed value ready for use by ECALL when control is eventually returned to ECALL.

The actual arithmetic is accomplished by using a software floating point package. This consists of a decimal to binary floating point input conversion routine, a binary floating point to decimal output conversion routine and a floating point interpreter for controlling the multiplication, division, addition, subtraction and exponentiation operations.

3.7. Pushdown List Controls

The PDP-8 has no hardware stack management to allow for recursive subroutine calls and therefore the FOCAL interpreter includes a software equivalent known as a pushdown list. The controls for the pushdown list allow for various operations such as saving data or subroutine return addresses on the stack and the inverse operation as shown in Fig.11.

3.8. Execution of a Program

Each stored line within a FOCAL program has within it a pointer to the position in core of the next higher numbered line. These pointers are used to control the sequence of operations within the program.

When execution of a program has been started, unpacking text pointers are set up to access text from the starting line. Once that line has been completed, the pointer at the beginning of the line is used to reset the text position to the start of the next line in the sequence.

If a program branch occurs, i.e. a GOTO command or a DO command, the line number in that command is decoded and used to determine the position in core of the required branch line. Operation then proceeds from that point in the normal fashion until the end of the program is encountered or an error condition is detected.

<u>Calling Sequence</u>	<u>Operation</u>
PUSHA	Saves the accumulator in the next free location of the pushdown list
POPA	Restores the last entry on the pushdown list to the accumulator
PUSHF ADDRESS	Saves three successive data locations starting at ADDRESS on the pushdown list. The data is generally floating point data , although it may be sets of pointers such as text pointers eg. TEXTP
POPF ADDRESS	Restores the last three entries on the pushdown list to the three successive data locations starting at ADDRESS
PUSHJ ADDRESS	A subroutine call in which the return address is saved on the pushdown list . ADDRESS is the starting address of the required subroutine
POPJ	Used in conjunction with the PUSHJ call . Uses the last entry on the pushdown list as the return address from a subroutine

Figure 11 : Pushdown List Controlling Instructions

<u>Program instruction</u>	<u>Best case</u>	<u>Worst case</u>
SET A =123.4	15 m secs	25 m secs
SET A=B	8 m secs	30 m secs
SET A=B*C	11 msecs	50 msecs

Figure 12 : Timing of FOCAL Commands

3.9. Error Diagnostics

When an error condition is detected, either in syntax or in an arithmetic operation, e.g. division by zero, a subroutine jump to the error recovery routine is forced.

This error recovery routine prints out the particular line number where the error occurred, and uses the subroutine return address, decoded from 12 bits and printed as a line number, as an error code. The error code can then be looked up by the user in a table of supplied error codes so that the exact position and nature of the error can be found.

3.10. Editing Facilities

Once an error has been detected and located, the powerful editing facilities of the FOCAL interpreter may be used for correction.

The MODIFY command can be used for correcting any part of any line within the program. This operates by copying the parts of the line which are correct on to the end of the text buffer, characters may be added or deleted by using the normal controls of the MODIFY command. This provides two lines with the same line number, the old one of which is deleted, the space recovered, and the new line linked into its correct sequence.

3.11. Control Pointers for FOCAL Programs

Most of the pertinent information dealing with the state of a FOCAL program is kept as a parameter set within the first half of page 0 of field 0, thereby making them accessible to all parts of the FOCAL interpreter. These parameters include the state of input and output text pointers, pushdown list pointer, floating point accumulators, variables and text pointers and character buffers.

3.12. Speed of Operation of a FOCAL program

The relatively slow speeds of execution of FOCAL programmes can be mainly attributed to the manner in which variables are stored and accessed and also the use of software routines for arithmetic operations.

For example, the execution of a single floating point operation can take in the region of 3 to 15 μ seconds because of the conversions necessary from decimal to binary floating points.

Fig. 12 shows the best and worst case execution times of simple FOCAL statements. The difference in times taken for the same command is dependent upon the position of the variable in the symbol table. As each variable is encountered in the statement, the variable search routine (GETARG) is used to locate the position of that particular variable. The search always starts from the beginning of the table; more time will therefore be required to locate the variable if it is situated near the end of the table.

Little timing advantage is gained even if constants are expressed as numbers rather than assigning that value to a particular variable name. This is due to the fact that these decimal figures have to be decoded from text each time they are used with the decimal to binary conversion routine.

Another reason for the slow execution of FOCAL can be attributed to text searching routines. If a program operates sequentially from its lowest numbered line through to its highest numbered line, excess timing overheads are not incurred as sequentially numbered lines are linked by pointers. If however a program branch is forced from a DO or a GOTO command, the interpreter must search the text buffer from the beginning in order to determine the position of the specified line before program execution can continue.

Most of these timing overheads can be significantly reduced (52) by adhering to a few simple programming rules. For example:-

1. Those variables which are going to be used most often should be defined first so that they are placed at the head of the symbol table.
2. Those lines of FOCAL which are to be used most often should be given lowest numbered lines so that they will be encountered early on in text search routine.

3.13. The Interrupt Processor

The FOCAL interrupt processor operates using a skip chain type structure, i.e. testing if a particular device has caused the interrupt, servicing it if it has and moving on to another device if not.

As FOCAL is intended as an off-line system, the devices in its skip chain includes teletype reader, keyboard and high speed reader only. The interrupt processor is primarily used for the character input and output from and to a terminal device.

3.14. Character Input and Output

Character input and output routines for use with a teletype terminal device are interrupt driven, thereby allowing data processing to continue while transmission to and from the teletype device is being completed.

Input of characters from a keyboard device is in general a slow process. As the computer is normally awaiting data before proceeding on to another stage of an operation the data will be processed immediately, thus requiring only limited buffering space. The FOCAL interrupt processor uses a single word for an input buffer which is used to store the data temporarily between being received and being used by the interpreter.

Data output is however a different process and it is preferable if it can be done with as little hold up in processing as is possible. This is accomplished in FOCAL by having a 16 character long ring buffer operating on a first-in first-out basis, Fig.13. Characters are loaded into the buffer by the interpreter under program control. The buffer is then emptied from the interrupt processor each time the output device signifies its readiness to receive another character.

This type of procedure will reduce the possibility of the system becoming output bound, i.e. the system waiting for the output device to become available, although, with such a limited size of buffer, the problem cannot be eliminated.

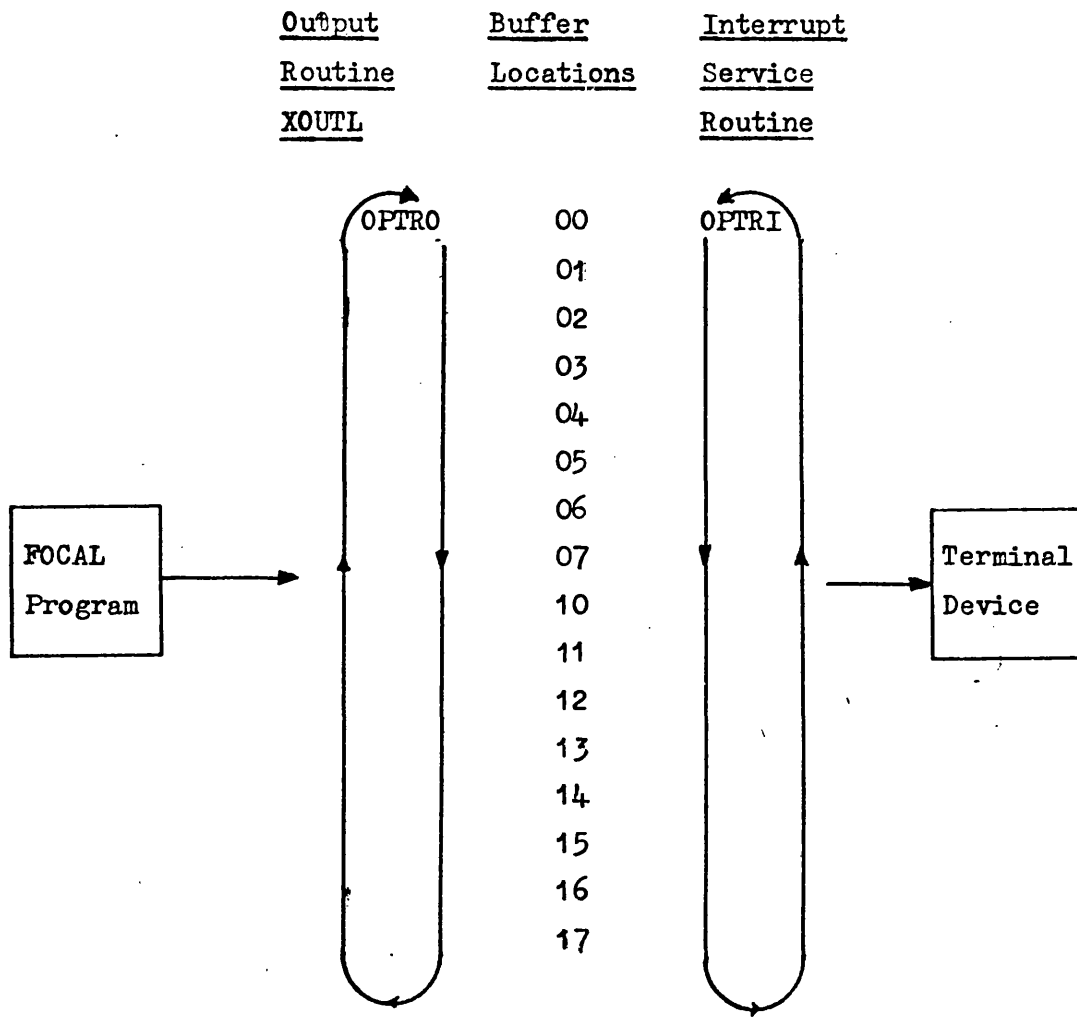


Figure 13 : Operation of Output Ring Buffer

CHAPTER 4.

Initial Modifications to FOCAL

4.1. Reconfiguration of FOCAL Software

FOCAL /69 was designed to operate in 4K of memory, Fig.6. and by using a standard overlay could be extended to use an additional 4K of memory, Fig.7. The 8K configuration, although allowing users to develop and execute quite large programs, leaves very little free space for system modification. Admittedly, two areas of core are shown as being reserved for paper tape loaders but this would only have released one page of core which was insufficient for all the proposed system modifications and enhancements.

Creating sufficient free core area within the FOCAL interpreter to accommodate enhancements to the system, therefore necessitated the reduction of the core area allocated to program storage in some manner. For example, new functions could have been located in field 1 by reducing the amount of that field allocated to the text. Alternatively new functions could have been added below the pushdown list in field 0 by reducing the core allocated to variables and pushdown list. A third method could have been to relocate part of the interpreter, such as the floating point package, in field 1 thereby releasing space in field 0 and reducing the area available for text.

All of these methods would have involved the generation of cross field linkages between the original interpreter and the additional modifications. They would also have rigidly defined storage allocated to text and variables, providing no simple method of implementing a facility whereby a user could trade off text storage for variable storage and vice versa.

A solution which was adopted, was to modify the interpreter so that the variables (53) and pushdown list were relocated underneath the text in field 1. This method released space in field 0 enabling extensions to be made to the interpreter without the need for cross field linkages.

Moving the variables and pushdown list storage areas into the same field as the text storage area, created the problem of the division of available storage area between the competing requirements of text, variables and pushdown list.

Fixed areas of core could be allocated to text variables and pushdown list as in the standard 8K version of FOCAL. This type of approach has the advantage that if modifications are made to either text or

variables then the other is not altered. It does however suffer from the disadvantage of being inflexible and not allowing users to trade off text space for variable space and vice versa.

An alternative to this approach would be to dynamically allocate the storage as in the original version of 4K FOCAL. This particular system starts the variable storage immediately after the text storage; the pushdown list builds back up from the end of the allocated area for storage and an overflow error condition occurs if either variables encroach into pushdown list space or vice versa. This method does have the advantage of using the available core storage to its maximum but has the disadvantage that existing variables are erased from the store if any of the text is modified.

A method which would overcome these disadvantages would be one in which a user could select for himself the amount of core to be allocated to text and the amount to be allocated to variables and pushdown list. The starting point of the variables list could then be changed at any time so as to accommodate either more or less text, with no possibility of text modifications erasing the variables list. This method would also allow a form of rudimentary chaining of FOCAL programs as text could be read in without disturbing data and data could be read in without disturbing text.

This method was therefore adopted for use in 8K FOCAL. An extension to the existing LIBRARY command was developed to enable the core allocated to text and variables to be selected by individual users. Fig.14. shows examples of the use of the modified LIBRARY command.

4.2. Saving and Editing Variable Tables

In a real-time situation requiring a moderately high data acquisition rate it may not be possible to process data at the same time as it is acquired. Data may therefore have to be accumulated for processing at some later time either using an off-line program or possibly another computer.

In order to accommodate relatively large programs in the limited space available within the minicomputer system, the facility whereby program segments could be "chained" together would be necessary. Program segments could then operate on variables created by a previous program

*L User types command character and carriage return
at the terminal device

0100 Start of text

3000 End of text

4000 Start of variables list

4000 End of variables list

:3400 The : prompt is echoed by the computer to signify
? 13.44 that it is waiting for the input of a four digit
* octal number terminated with a carriage return.
Any other reply will cause the command to be
aborted

The error code 13.44 is printed to signify that
the command has been completed. The system returns
to command mode

*L Repeating the command shows that the start of the
variables list has been relocated

0100

3000

3400

3400

:A Command aborted by use of an illegal character
and the system returned to command mode

*

Figure 14 : An Example of the use of the Modified LIBRARY Command

segment. Having arranged text and variables in the manner described, modular programs can be accommodated by loading the various segments without disturbing the variable list.

In both these situations it would be advantageous if the data tables could be edited and unwanted variables deleted. The remainder of the data table could then be saved either as a hard copy for processing on another machine or within the existing variables area for use with the subsequent program segment.

4.2.1. Editing Variable Tables

An extension of the ERASE command was therefore developed and included so that a named variable or string of variables could be deleted from the symbol table.

The command is of the form

E F, A, B, C.

where the F is used as a switch to the extension of the ERASE command using what was previously an error exit. The rest of the characters are those signifying the variables to be deleted from the symbol table separated from each other by delimiting commas.

The command operates by overwriting the named variable with the last variable in the list, reducing the end of list pointer by the number of words occupied by a variable hence recovering the available space. It may be used as either a direct or indirect command.

4.2.2. Saving Variable Tables

Saving a symbol table could have been accomplished by adding a binary punch routine to FOCAL so as to enable a symbol table to be punched out in standard binary format. This would have the advantage of speed on punching and re-reading although it would necessitate a modification of Binary Loader so that :-

- (a) It could be operated from a remote console
- (b) The data could be read into any specified location within field 1. and not merely the position from which it originated.

An alternative procedure, which was eventually adopted, was to modify the existing symbol table dump routine, (T. \$) so that the variable names and values could be punched out in the format of a direct SET command. The tape produced would be in ASCII code and could be read into the computer in the same manner as that used for program tapes.

The existing symbol table dump routine prints a list of all variables held in the table in the form.

VA (XX) = 123.456

VB (XY) = 789.012

The subscript being printed as a two digit integer number between 00 and 99. Subscripts of greater value than 99 therefore are printed incorrectly, which is unsatisfactory when the data tape is to be read back in again.

This had to be overcome by using the floating point binary to decimal conversion output routine and avoiding the automatic printing of the "=" in that routine.

The direct SET command format was achieved by forcing an S and a space to be printed immediately before the variable name.

4.3. Input Buffer Overflow

An annoying feature of the 8K FOCAL system is that of the input buffer overflowing when paper tape programs and data tables are read in through the lowspeed reader of a teletype terminal. As users operating from remote terminals will only have a low speed reader device available, it was imperative that this problem be remedied. This was particularly true when modifications had been made to allow for the creation of ASCII coded data tapes.

The problem occurs because the lowspeed reader is operated under interrupt control and not under program control: Characters can therefore be presented to the computer from the teletype at a rate which is independent of the rate at which the characters are processed. In the event of a second character being presented before a previous character has been processed, an error condition results and the corresponding error code displayed at the terminal device.

The problem could be overcome by including a non-interrupt driven input routine, which could be selected from the terminal prior to the reading in of punched paper tapes. Once the tape had been read in, the interrupt driven input routine could be reselected.

An alternative procedure would be to reduce character processing time. This could be achieved by preventing input characters from being echoed on the terminal printer device, thereby saving valuable processing time and also avoiding the possibility of holding the system up in an output wait loop.

This second alternative was chosen because of its ease of implementation. Fig.15 shows a flow sheet of the character input routine used by the interpreter for picking up characters for processing from the input buffer. The PRINTC subroutine call is used to load the input character into the output buffer for echoing. If the PRINTC call could be bypassed on request, the problem of input buffer overflow would be solved. Fig.16 shows the modifications made to the input routine in order to implement this facility.

If a user wants to read in a program or data tape through the low speed reader, a CTRL/X character can be typed in at the keyboard prior to reading the tape. The PRINTC call within the character input routine is replaced by a NOP instruction so as to eliminate character echoing. The echo facility can be restored by typing a CTRL/R character which has the effect of restoring the PRINTC call in the character input routine.

4.4. Enhancement to the MODIFY Command.

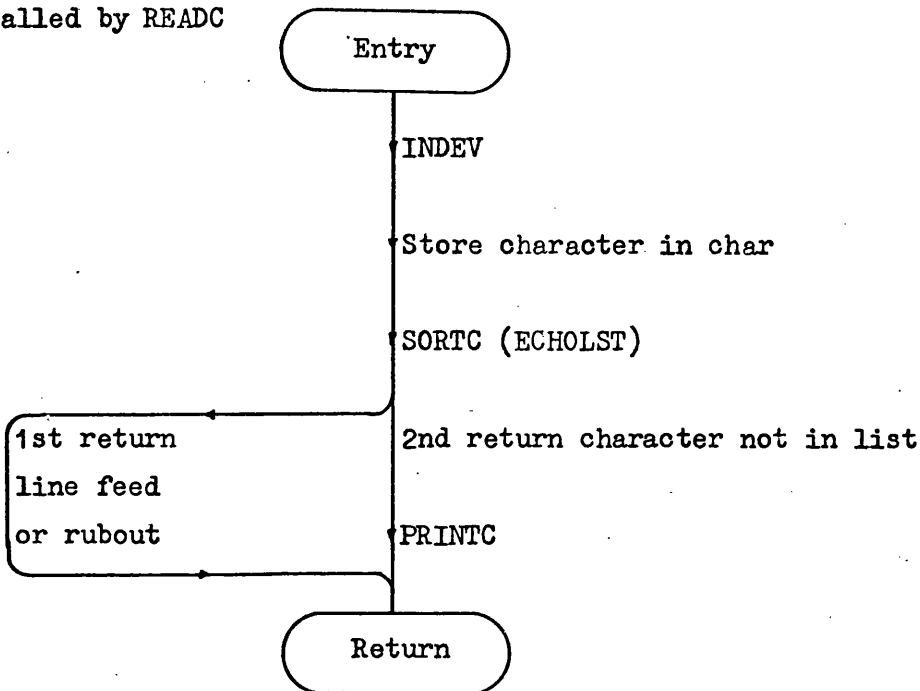
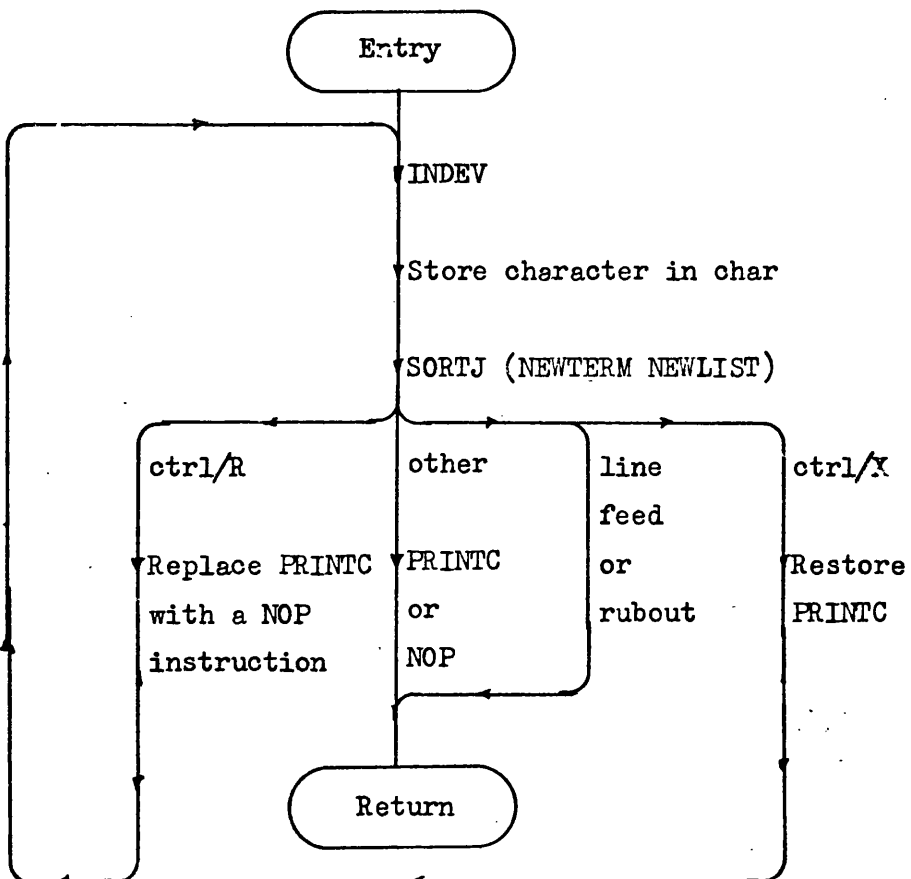
The existing MODIFY command allows a user to edit any line within an indirect program. There are however no editing facilities available for re-arranging the order of the lines in the program or for copying lines. Such a facility would be extremely useful particularly during the creation and modification of programs.

On examination it was found that such a facility could be included by adding a minor modification to the MODIFY command.

The MODIFY command operates by first searching through the text buffer for the specified line of the program. When found, a new line with

CHIN

Called by READC

Figure 15 : Original character Input RoutineFigure 16 : Modification to Input routine CHIN for Echo Suppression

the same line number is initiated at the end of the current text buffer. Typing in a search character causes the contents of the old line to be copied into the location of the new line until a character corresponding to the search character is encountered. Modifications are then made to the line and the line terminated immediately if a carriage return is typed or the rest of the old line copied if a line feed is typed. The old line is then deleted and the space recovered before finally linking the new line into its correct sequence by setting the pointer at the beginning of the line as described earlier.

If however, the new line was provided with a different line number from the original line, all the modifications would be added to the new line, the old line would not be deleted and the new line would be linked into its correct sequence.

Figs.17 and 18 are flow sheets of the original MODIFY command and the extension incorporated so as to provide for line copying.

The new MODIFY command has the same syntax as the old MODIFY command for line editing. However, for line re-arrangement the command takes the form

M nn.xx, oo.yy

where nn.xx is the required line number for the new line

oo.yy is the line number of the existing line

the comma is used as a switch to denote which type of command it is.

All the options available within the original MODIFY command still apply to the extended form. The modifications however are inserted into the new line leaving the old line unchanged.

4.5. Hard Copy Facilities for the ASK command

The existing ALTMOD reply in response to an ASK command is very useful. It does not however echo the value of the variable to the user.

Another problem encountered with this facility is that the ALTMOD key does not appear on some of the more recent terminal keyboards.

It was therefore thought advisable to alter this response so as to overcome these defects. The LINE FEED character was selected as being a suitable replacement for ALTMOD.

MODIFY. Line Editing Routine

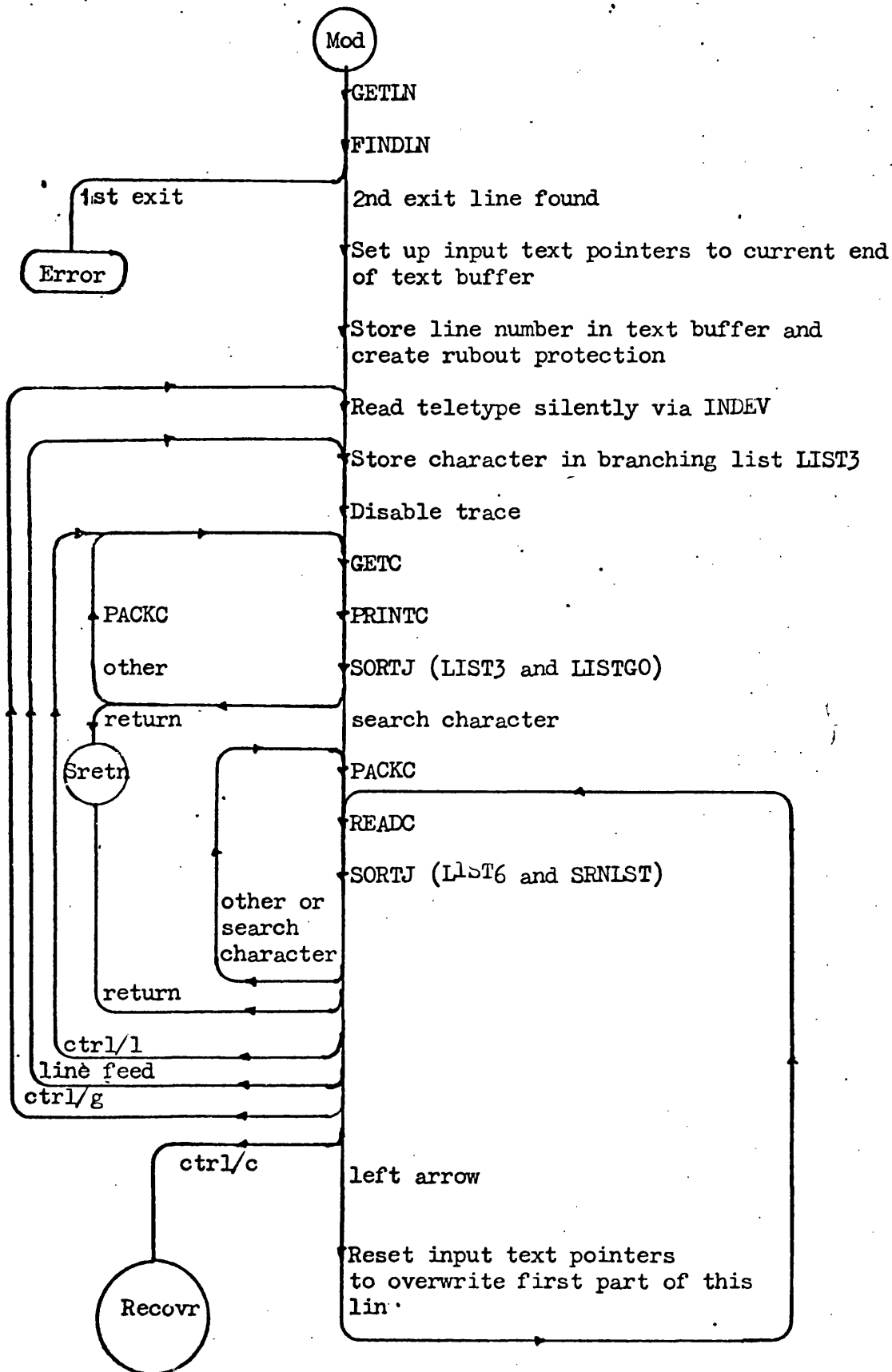


Figure 17. Flowsheet of the MODIFY Command

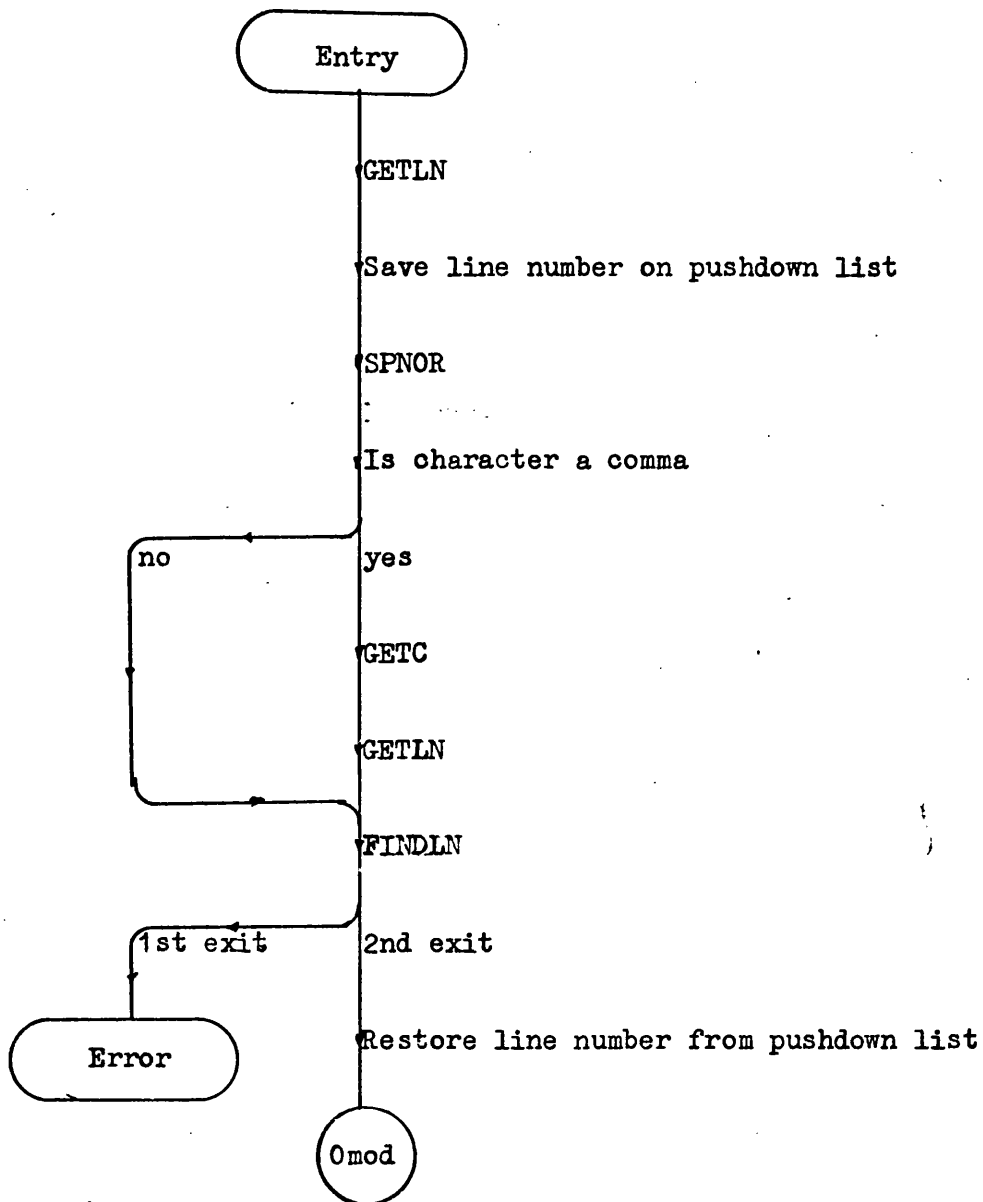


Figure 18 : Extension to MODIFY Command for Line Duplication

By responding to an ASK command with a line feed, the value of the required variable is obtained from the symbol table and loaded into the floating point accumulator. The binary to decimal output conversion routine is used to print the value of the variable on the terminal printed device. Program control is eventually returned to the ASK command leaving the value of the variable unchanged. This operation is as described in reference 54

4.6. Protection Systems

4.6.1. Command Buffer Overflow

When a character string is presented from a terminal keyboard device, it is initially entered into the command input buffer for the reasons explained earlier. In the original version of 4K FOCAL the command buffer was protected against overflow. In the 8K version of FOCAL/69 this particular facility was omitted. This was probably due to the difficulty of implementing the system with the different core configuration. The command buffer was therefore extended to accommodate about 100 10 characters which is greater than the number of characters which can be placed on a single teletype line.

In an effort to make FOCAL as secure as possible, it was thought that command buffer protection would be a desirable feature, particularly as overflow causes corruption of the system.

As the new system incorporated a text variables and pushdown list storage system similar in many respects to that encountered in 4K FOCAL, command buffer protection was implemented in an identical manner.

4.6.2. Power failure Protection

The central processor of the PDP-8 includes an optional power failure protection system, which causes a program interrupt if the input voltage level is reduced below a certain level. By including a power failure detection instruction in the interrupt processor skip chain active program registers may be saved in known locations so as to avoid corruption while the processor power is off.

When power is returned to the system, the processor is immediately started at location 0000 field 0 . Control can then be transferred to an automatic restart routine which will reinstate the active program register. Program operation can then be continued from the point where the interrupt occurred.

4.7. New Functions

A new random number generator function FRAN was included in the system, based upon the techniques explained in ref.55.

4.8. Summary

These modifications were implemented by using a PAL III Assembler available on the University's I.C.L. system 450 computer (56), a standard binary coded punched paper tape being produced as an overlay to 4K Focal 1969. The listings of the modifications have been produced on microfiche together with supporting documentation in the form of flowsheets which can be found in Appendix C. The 8K core configuration produced is shown in figure 19 and the system was released as 8K FOCAL EXTENSIONS.

The system enhancements provide users with an off-line conversational programming system which can be operated from remote terminals. The inclusion of software to handle power failures essentially means that the computer can be started and stopped from a single mains switch. There is no longer any need for terminating program operation on power down and restarting program operation after power up as the active register of the computer are stored and reloaded automatically. The system can therefore be used when required without the need for specialist personnel to initialise or halt the computer.

Approximately 800 locations of core are released within the same field as the interpreter, making this particular configuration a suitable starting point for further modifications to allow for real-time programming.

	8K Focal Extensions Field 0									
0	Zero Page Pointers									
1	Command Decoder					Getln				
2	Entabf		Do			Pushdown List Controls				
3	Control & Transfer		Write			Testc		Sortc		Input
4	Comlst		If		Set & For			Dys & Int		Conpro
5	Type & Ask			Modify			Sortj		Adc, Outl and others	
6	Getarg						Spnor, Testn, Ren, Porj and others			
7	Ecall									
8	Terms		Ern, Abs et al		Tstlpr, Partest		Delete			Reado
9	Erase		Findln			Getc			Endln	
10	Esp, Prntln, Prnt and Printc					Packs			Extensions	
11	Interrupt Processor, I/O Routines and Error Recovery Routine									
12	Rubout		Symbol Table Dump				C/P buffer		Pays, Cayn, Caub8	
13	Extensions to Library and Variable search					Echo disable and Power Failure Routines				
14	Fran									
15										
16										
17										
18										
19										
20										
21										
22										
23	Exp					Log			Atn	
24										
25	Cos and Sin								Mod and Altmde	
26	FLOATING POINT									
27	INPUT AND OUTPUT									
28	ROUTINES									
29						High Speed Reader Routine				
30										
31										
32										
33	FLOATING POINT INTERPRETER									
34										
35										
36	Sgt					Library				
37	Variable Erase Routine					Further extensions & Fan Switch				

0	8K Focsl Extensions Field 1									
1	Command Input Buffer									
2										
3										
4										
5										
6										
7	TEXT									
8										
9										
10										
11										
12										
13										
14										
15										
16										
17										
18										
19										
20										
21										
22										
23										
24										
25										
26										
27	VARIABLES									
28										
29										
30										
31										
32										
33										
34										
35										
36										
37	← PUSHDOWN LIST Loaders									

Figure 19

CHAPTER 5.

A Single User Real Time Programming System

5.1. Real-Time Programming

The basic requirements of a real-time operating system have been outlined in the introduction. In summary, these are:-

1. The system should provide users with timing facilities of two types:
 - (a) The ability to access or record current and/or elapsed time
 - (b) The ability to multiplex data, enabling data signals to be sampled or transmitted in a synchronous manner. This particular timing facility should also include some form of priority system whereby pre-selected signals can be sampled or transmitted in preference to other signals.
2. The system should provide users with a means of communication with peripheral devices through the computer interface system, thus allowing signals to be obtained from or transmitted to external devices.
3. The system should provide facilities for linking timing, mathematical and input/output functions for the purpose of control.
4. The system should provide users with the ability to communicate with a program which is being executed, enabling parameters used within the program to be altered without causing termination of program operation.

The development of the FOCAL interpreter as an on-line programming system therefore involved the integration of hardware driving routines into the interpreter structure. In order to accomplish this, it was necessary to be aware of:-

- (a) The hardware available and the methods by which data could be transmitted to or acquired from particular items of hardware.
- (b) Manipulation routines available within the interpreter structure so that data transmissions to or from the hardware could be provided for in a form acceptable

to the user utilizing as far as possible existing arithmetic facilities and character handling facilities.

5.2. The PDP-8 Computer and Interface System

The internal mode of operation of the PDP-8 computer, i.e. the manner in which instructions are decoded and used by the internal registers is described fully in reference 58. Fig.20 is a schematic diagram of the PDP-8 configuration showing all the major internal registers of the system and the flow of data to and from them.

5.2.1. Device Selection and Channel Addressing

The associated interface system is based upon the Kent K70 interface (36) and is used for the timing, buffering and control of data transfer to and from the processor accumulator. The transfers are performed in response to an input/output transfer instruction (IOT instruction, octal code 6) issued by the computer program.

The issuing of an IOT instruction causes one or more input/output pulses, used for flag operations and to initiate data transfers, to be sent to the device selectors within the interface. Each device selector is also connected to an appropriate binary pattern of bits 3 through to 8 of the memory buffer, so that it is activated in response to the appearance of its own code in the memory buffer.

For example, the digital input system has been allocated the device code of 17. To read the status of an external contact into the accumulator from the digital input system, the octal instruction 6172 is issued from the program. The first digit, 6, is used by the internal register of the computer to denote that an IOT instruction has been issued. The second and third digits are used by the device selectors to determine which particular I/O device is required. The fourth digit is used to generate the IOP pulse required for reading the status of the external contact into the accumulator.

This method of device selection would allow up to 64 peripheral devices to be selected if some of the device codes were not used for internal options in the PDP-8 and for standard peripheral devices. For instance power failure protection, field operations and interrupt

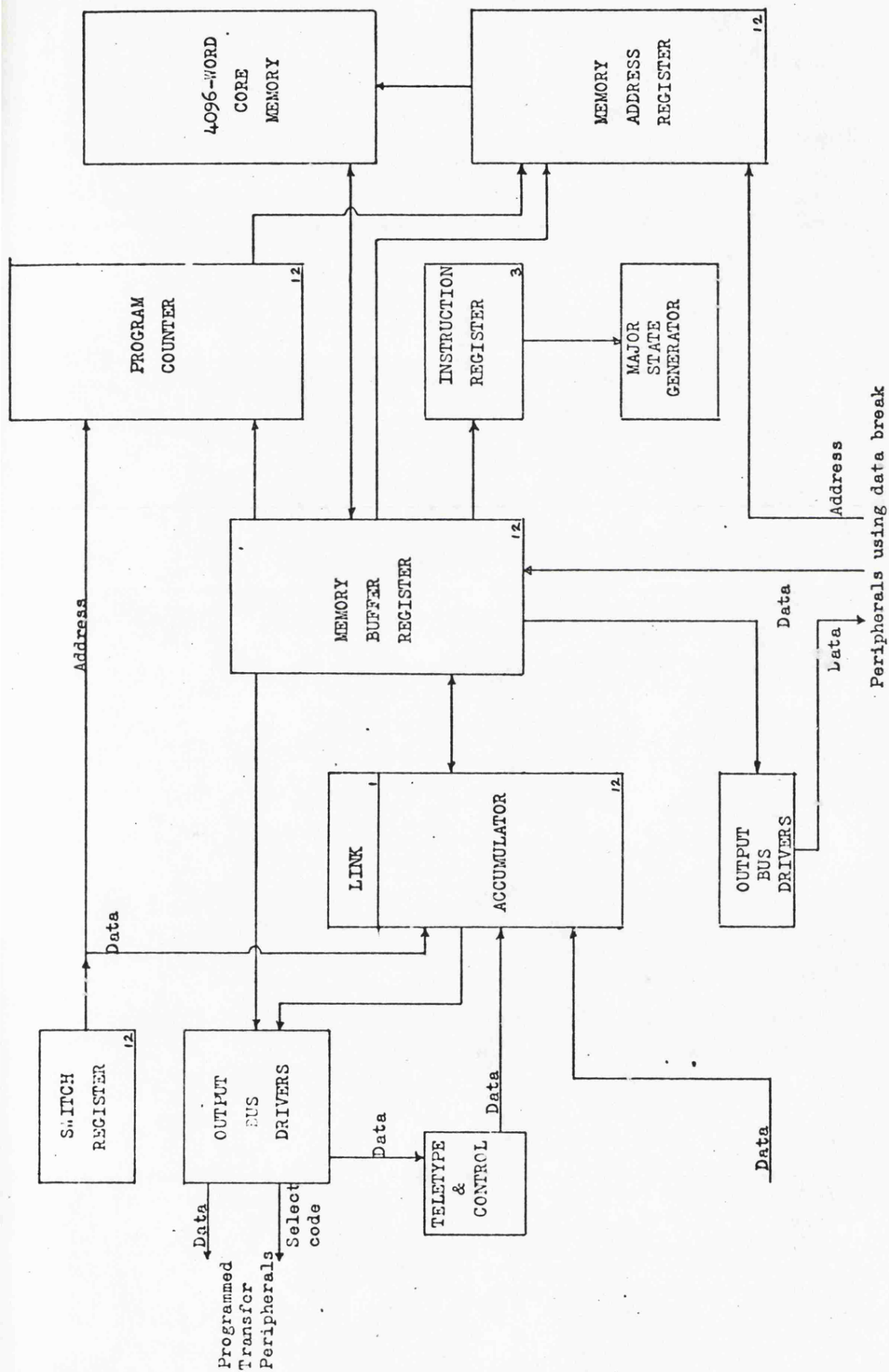


Figure 20. PDP-8 Major Register Block Diagram

manipulation require IOT instructions and device codes. Standard peripherals such as magnetic storage devices, photoelectric reader, data for which does not pass through the K70 interface system, require the use of other device codes.

The number of device codes available for data transmissions through the interface is therefore limited. In order to extend the I/O capabilities of the computer, the interface has been provided with a channel address selection system. This allows more than one of the same type of peripheral to be individually addressed from the same device selector.

The channel address of the device required is loaded into the accumulator and transferred to a 6 bit output buffer register under program control. Binary to octal decoding of the register is accomplished by hardware to give outputs on a 16 line address highway for the selection of the required device. The IOT for the device can then be issued so as to provide either a data transfer to or from the device or to determine the status of the device.

5.2.2. Input Devices

The following input devices are linked to the central processor unit by the K70 interface system.

1. A successive approximation analogue to digital converter as an integral part of the central processor. Solid state addressable single pole scan switches are allowed to time share the converter. An input signal in the range 0 to 5v produces a 0 to 10 bit value in the processor accumulator. The conversion time is approximately 40 μ seconds.
2. A digital input system for the detection of on-off or fleeting contact closures. The inputs are arranged in groups of four, each group of inputs being address selectable. Isolation between the plant equipment and the computer is provided by interposing relays.
3. An input counter card system for accepting inputs from transducers with a frequency modulated output. The 12 bit input counters are addressable and are read and reset by program.

4. An extension of the input counter card system for accepting binary serial data transmitted from a digital panel meter (59). The serial data pulses from all digital panel meters are read into the input counter card on channel address 0 when requested by program. Each panel meter is also individually addressable. The conversion of an analogue signal in this fashion takes approximately 60mseconds.

5.2.3. Output Devices

The following output devices are linked to the central processor unit by the K70 interface system:-

1. A pulse width modulation system, capable of providing for the output of data generated by program for the purposes of control and display. A seven bit output word comprising of six data bits and one sign bit can be sent to one of the addressable counters under program control. The data could represent for example, the change in position of a control actuator calculated by an incremental control algorithm. An output voltage signal is then output on one of two lines, depending upon the sign, until the counter has been decremented to zero by an external clock.
2. A digital output system for switching purposes. The outputs are arranged in groups of 4, each group being address selectable by program. A four bit output word set by program is used to select the status of each group of 4 changeover relays.
3. An alarm output system for signalling the occurrence of a fault condition to the user. The alarms are arranged in groups of four, each group being address selectable. A four bit output word is used to select the status of each group of four alarm annunciators. Individual alarms are set by providing that particular alarm with a 0 from the accumulator. Setting the respective bit in the accumulator to a 1 will clear the alarm signal.

4. A digital to analogue conversion system for output to analogue recording devices. The 9 bit converter is loaded from the accumulator under program control and the converter output is then held on one of the addressable set and hold amplifiers. The conversion process takes approximately 200μ seconds providing an analogue output signal in the range 0 to 5v with a maximum drift rate of less than 5 milli-volts per second.

5.2.4. Other Peripheral Devices

Other peripheral devices available on the computer system are:-

- i. A high speed photo electric reader, capable of reading punched paper tape at a maximum rate of 500 characters per second.
- ii. A high speed paper tape punch operating at a maximum rate of 50 characters per second.
- iii. Two standard ASR 33 teletype units.

Figure 21 shows a schematic diagram of the computer system and its associated hardware.

5.2.5. Interrupt System

The standard PDP-8 system allows any external device connected to the interrupt bus to interrupt the program at any time. This causes a temporary halt so that the interrupt may be serviced. In addition to this, the interface provides a four level priority interrupt system capable of being enabled or disabled at any level by program. External devices are hard wired to a particular priority level.

5.2.6. System Clock

The interface provides a real-time clock driven by the 50 cycle mains frequency protected against spurious pulses by a phase locked loop. The clock, which is essential to on-line programming, is connected to the interrupt bus and generates interrupts at fixed time periods. The period at which the clock provides interrupts is switch selectable in the range of 10μ seconds to 1 second.

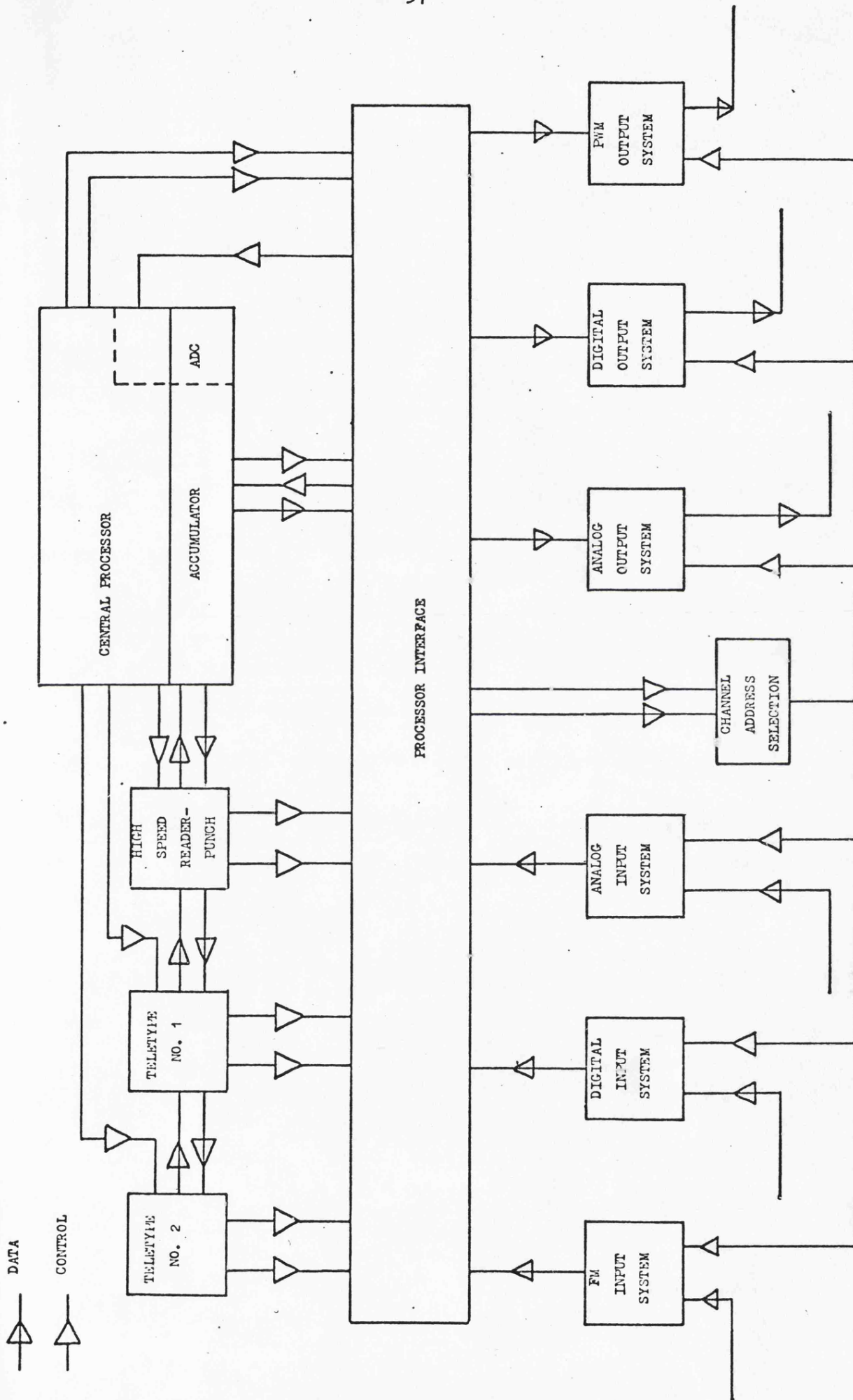


Figure 21. System Hardware Block Diagram

5.2.7. Fault Protection Systems

The computer system has been provided with limited fault protection in the form of a power failure detection system. As described previously, the detection of a low voltage condition, causes a program interrupt which can be serviced within an interrupt processor so as to save active program register.

The pulse width modulation output system (output counter cards) is designed such that it will fail safe. Additional protection against spurious outputs and timing errors is provided by the "watchdog timer". This disables the output of the sign bit of the signal from the output counter card if the timer is not updated at regular 1 second intervals. It is necessary to include a routine within the interrupt processor for resetting the timer every second. In doing so, not only is the output system protected, but a visual display is obtained as to the correct operation of the clock.

In order to assess the accuracy of the analogue to digital converter, standard signals have been provided on channel address 0 and 1. These signals can be read in by the user and compared with known values.

A similar technique has also been employed on the digital panel meter sub system.

5.3. Real-Time Software

The following section discusses the methods available for adapting FOCAL to provide a real-time operating system. The reasons why a particular method was adopted are described and the ways in which they can be used are illustrated.

The modifications were accomplished by using the PAL III cross assembler available on the University ICL system 450 computer. The modifications produced for 8K FOCAL Extension were also included so as to produce a binary coded paper tape overlay for 4K FOCAL 1969. The listings have been included on microfiche and supporting documentation in the form of flowsheets have been included in Appendix D.

5.3.1. Current or Elapsed Time

The real-time clock available in the computer interface provides program interrupts at a switch selectable rate. There are, however, no external hardware registers available for maintaining a record of current

or elapsed time. If a record of current time was to be kept, therefore, a service routine must be included within the interrupt processor in order to count the number of clock interrupts. Other routines must then be incorporated into the interpreter structure so that the clock counters may be accessed by the user.

This could have been achieved in either of two ways:-

1. By summing the clock interrupts received in a multiple word counter, i.e. for a clock interrupt rate of 100 per second, a single 12 bit word would provide an elapsed time record of approximately 0.7 minutes. Therefore a 36 bit counter would be required to extend the elapsed time register into days. The elapsed or current time record in terms of days, hours, etc., would then require the decoding of the counter upon a request from the user.
2. By providing a series of counters for fractions of a second, seconds, minutes, hours and days. Each clock interrupt would be used to decrement the count in the fractions of a second counter. When a counter has been reduced to zero it would be provided with a reset value and cause the next higher counter to be decremented by one unit. Thus a continuous record of time since initialisation would be available for access by users.

The first of these methods although requiring less store and time in the interrupt processor would have required more handling routines in order to decode the total interrupt count into absolute terms. The counter would also require resetting each time the clock interrupt rate was changed, unless a separate counter was included for counting interrupts per second as in the case of the alternative procedure.

The second method was therefore chosen for implementation within the interrupt processor of FOCAL. The counters were made accessible to users by the use of a new function FTIM. By using the FTIM function as shown in Fig.22, users can access any one of the four counters available, e.g. `S Z = FTIM (0 MNS)` would set the value of Z to the current value of the minutes counter.

It has also been arranged that the counters may be initialised to any value by use of a multiple argument variation of the FTIM function, for example, `S Z = FTIM (0 ST, A, B, C, D)` initialises the clock counters

```

C-SK REAL TIME FOCAL @1974 WITH DYN

21.01 A "TIME PLEASE",1,"SECONDS",S,1,"MINUTES",M,1
01.02 A "HOURS " ,H,1,"DAY " ,D,1
01.03 S Z=FTIM(OST,S,M,H,D);D 3
01.04 D 2
01.05 I (FTIM(OST,S,M,H,D))1.04,1.04
01.07 I (TI)1.09
01.08 G 1.05
01.09 S TI=0;G 1.04

02.01 S S=FTIM(OST,S,M,H,D);S M=FTIM(OST,S,M,H,D);S H=FTIM(OST,S,M,H,D)
02.02 S C=S+100*M+10000*H;T %06.0,C,1
02.03 F I=1,500;S V=1
02.04 C LAST LINE IS A TIME DELAY

03.01 T " H M S",1
*G
TIME PLEASE
SECONDS:00
MINUTES:25
HOURS :09
DAY :18
H M S
= 92500
= 92600
= 92700
= 92800
= 92900
= 93000
= 93100
= 93200
= 93300

```

Figure 22. Use of the FTIM Function

to the values of seconds, minutes, hours and days designated by the arguments A, B, C, D respectively. The OST argument had to be included as a switch to provide this facility. The variable assigned on the left hand side of the set command is used as a dummy variable for the function call, and has no particular significance.

5.3.2. Synchronous Data Scanning

In providing the system with facilities for synchronous data acquisition, it was necessary to determine the effects errors in timing have on data acquisition and signal reconstruction. The need to provide precise timing would not only affect the methods by which a synchronous sampling scheme could be implemented but would also affect the modes by which data could be acquired from peripheral devices.

A simulation study of the effects of errors in timing on signal recovery was therefore made and the results compared with theoretical solutions (70, 71).

The method results and conclusions are described fully in Appendix B. In summary, the results of the simulation study agreed with the theoretical work in that the maximum amount of error which can be tolerated is dependent upon the maximum frequency content of the signal being processed. It was also concluded that for a system which is essentially going to be used for relatively slow data rates, synchronous sampling could be set up by use of clock flags accessible to the user from functions in the high level language. Input and output transmission to peripheral equipment could then be performed directly without the use of clock driven handlers.

It was therefore arranged that a user may define three scanning flags within the interrupt processor. Counters within the interrupt processor are set by the user and decremented by one unit every clock interrupt. When a counter has been decremented to zero, a software flag associated with the particular counter is set, and the counter reset to its initial value. Scan flags are cleared either by the next clock interrupt if they have not been accessed by the users program, or by accessing the scan flag from the program.

A new function has been included in FOCAL to perform these operations.

The scan flags must be initially set up by defining the arguments of a multiple argument FLAG function in the following manner:-

4.01 S N = 4; S A = 1; S B = 2; S C = 4

4.02 S Z = FLAG (OST, N, A, B, C)

The OST argument is used as a switch to denote that the remaining arguments are to be used for setting up the scan flag counter.

N is an argument used to denote the number of interrupts generated by the clock every second. In the above example, the clock will have been set to 4 interrupts per second.

The arguments A, B and C define the periods at which the scan flags will become set in terms of number of clock interrupts. In the above example, the scan flags A, B and C will therefore appear at $\frac{1}{4}$ second, $\frac{1}{2}$ second and 1 second intervals respectively.

Interrogation of the scan flags has been implemented by means of a single argument FLAG function. If the scan flag being interrogated is found set, the floating point accumulator is set to a negative value, thus making it possible to use the IF command in FOCAL for interrogation, e.g.:-

10.01	I (FLAG (A))	10.03 , 10.05 , 10.05	
10.03	DO 11		$\frac{1}{4}$ second task
10.05	I (FLAG (B))	10.07 , 10.09 , 10.09	
10.07	DO 12		$\frac{1}{2}$ second task
10.09	I (FLAG (C))	10.11 , 10.01 , 10.01	
10.11	DO 13 ; G 10.01		1 second task

The negative value of the floating point accumulator causes the first exit of the IF command to be taken if the software scan flag is found in a set condition.

The FLAG function makes use of the sort and branch routines available in FOCAL, the argument of the single argument function being tested against a list of possible flag values. If a match is not found, an error condition results. If a match is found, then the current value of that particular scan flag is obtained. This structure will therefore not allow two flags to be set to the same period, as the first one will always be found as the matching value in the table.

As a certain degree of error is acceptable in synchronous sampling (Appendix B), a synchronous sampling scheme could be achieved by using the FTIM function by a differencing process for example:-

```

8.01  S   Z1 = FTIM (0SCS)           Initialisation
8.03  S   Z2 = FTIM (0SCS)           )
8.05  I   (Z2-Z1-1) 8.03 , 8.09 , 8.99 ) Timing wait loop
8.07  I   (Z2+59-Z1) 8.99 , 8.09 , 8.03 )
8.09  S   Z1 = Z2                     )
                                           ) Operation
8.40  G   8.03                       )
                                           )

8.99  T   "TIMING ERROR" , !

```

Approximate timing errors of 40 to 50 m seconds would be incurred by the evaluation processes needed in such a system. However this would be acceptable for slow data rates.

5.3.3. Program Priority System

A certain degree of priority structure could be achieved with the FLAG function with suitable program organisation. In the example shown below, FLAG A is interrogated after the completion of any of the tasks associated either with FLAG A or either of the other two flags. This type of structure would increase the possibility of the task associated with FLAG A being executed in preference to any of the tasks associated with the other flags.

```

10.01  I   (FLAG (A))      10.03 , 10.05 , 10.05 )
10.03  D 11 ; G 10.01      )
10.05  I   (FLAG (B))      10.07 , 10.09 , 10.09 )
10.07  D 12 ; G 10.01      )
10.09  I   (FLAG (C))      10.11 , 10.01 , 10.01 )
10.11  D 13 ; G 10.01      )

```

This arrangement is however of no use in a situation where batch data from an experiment run is being processed at the same time as the experiment is being controlled. This situation requires that the processing be terminated whenever a control scan is necessary and would be impossible with the existing flag structure.

5.3.3.1. FOCAL Break Points

In general, FOCAL's internal subroutines are non re-entrant. However, once a line of text has been executed the routines can be used again without the need for saving subroutine return addresses and parameter used within the routine. There are therefore two convenient break points within FOCAL where a running program may be held up, a different section of the program executed, and then return to the original FOCAL sequence.

One of these points is at the start of each line or sub line (i.e. a section separated by a ;terminator) and the other is at the end of a complete line before control is transferred to the next FOCAL line. The second of these two break points requires only the saving of a single parameter, the pointer at the start of the current line used for pointing to the next line. The first point requires the saving of text manipulation pointers in addition to the saving of the pointer to the next line.

At one of these break points, it would be possible to temporarily hold the operating sequence of a FOCAL program in order to check the status of some external trigger or clock flag. If this was found in a set state, the pointers of the operating background program could be held, a separate foreground section of the program executed, eventually restoring the program pointer of background program so that execution could continue. The foreground program could be implemented by forcing a DO command on a particular group of statements as shown in reference 44.

For reasons of simplicity, it has been arranged that at the end of every line, a software flag set every second in the interrupt processor, should be interrogated. When the flag is found in a set condition, the execution of group 31 commands has been forced by use of a DO subroutine call before continuing on to the next line in the original sequence. If the flag is not set then the background program will continue in the manner prescribed by the sequence in the FOCAL program.

This arrangement allows an essential foreground task to be executed in preference to a background task which takes up any spare time available. It can also be used as a priority tasking system and an example is given in Fig.23. It's main disadvantage becomes apparent if the task to be executed is longer than 1 second, in which case group 31 will be executed at multiple levels which the system is never able to complete.

```

10.01 I (FLAG (A)) 10.10
10.03 R
10.10 S M = 1
10.12 S X2 (M) = FIN (MV (M) , CH (M))
10.14 S DP = KG(M) * ((X1(M)-X2(M))+A * (SP(M)-X2(M))/(N * IT (M))))
10.16 S OT (M) = OT (M) + DP
10.18 I (FABS (OT (M)) - 512) 10.22 , 10.22
10.20 S OT (M) = 512 * FSGN (OT (M))
10.22 S X1 (M) = X2 (M)
10.24 S M = M + 1 ; I (M-MX) 10.12 , 10.12
10.26 R

```

```

31.01 S MM = 1
31.03 I (50 - FABS (OT (MM))) 31.07
31.05 S OR = OT (MM) ; G 31.10
31.07 S OR = 50 * FSGN (OT (MM))
31.10 S OT (MM) = OT (MM) - OR
31.12 S QA = FINC (CO (MM) , OR )
31.14 S MM = MM + 1 ; I (MM-MX) 31.03 , 31.03
31.16 R

```

NOMENCLATURE

A	=	Number of clock pulses between appearances of flag A.
M , MM		Counters for control loops
MX		Total number of control loops
X2 (M) X1 (M)		Current measured value and last measured value of control loop
MV (M)		Input device required
CH (M)		Channel address of input
KG (M)		Gain factor of control loop
SP (M)		Set point of control loop
IT (M)		Integral action time of control loop in seconds
N		Number of clock pulses per second
OT (M)		Summed output values
DP		Temporary stores
OR		
CO (M)		Output channel address

Fig.23 : PCI Control Algorithm Written in FOCAL

It was found necessary to include a means of enabling or disabling the foreground - background facility from the users terminal. If the system is constantly enabled, an error condition is produced if no lines for group 31 are included in the program. Although this is easily remedied, it provides a constant source of annoyance to forgetful users. If the facility is disabled by a patch in the software, it necessitates an experienced user inserting in the required single word into core so as to re-enable the system. This is undesirable and would be better if each user could enable or disable the system if and when required.

This was accomplished by an extension to the ERASE command, which provides a suitable error exit that can be used to branch to various commands, executable in direct or indirect mode. Thus a user may enable the foreground - background routine by typing E C at his teletype console and disable the routine by typing E D at his console. The C and D are detected by the extension to the ERASE command and are used to either insert or clear a POPJ call in the foreground - background routine.

5.3.4. Input/Output Transfers

5.3.4.1. Clock Driven Peripherals

Having decided upon the method of synchronous scanning and that peripheral devices could be accessed directly (Appendix B), the question arose as to whether any of the peripherals would have to be driven by the clock interrupt system.

This is a question of whether a particular task can be accomplished more effectively and conveniently if done automatically rather than a question of synchronous timing errors or delays in conversion time if the task is done when requested.

It was found necessary to include service routines for the input and output counter cards within the clock service routine.

(a) Input Counter Cards

An input counter card is a device capable of counting input pulses in a 12 bit counter. Each time the count is read, the counter is reset to zero and the counting procedure continued. If the counter is not read at a sufficiently high rate, the counter will eventually "roll over" to zero, resulting in an erroneous input count. To avoid this possibility, it was essential that the input counter cards be read at fixed time intervals, the

period of which should be small enough to prevent roll over.

Another reason for driving the input counter cards from the clock routine is that it is generally the rate at which pulses are input which is of interest. It is therefore essential to know the exact time interval between successive readings of the input counter card. It was however found to be more easily and accurately accomplished by reading automatically at fixed time intervals.

It was therefore arranged that the counter cards on channel addresses 1 through 6 (0 being dedicated to the use of panel meters) should be read once every second. The values obtained being stored in an input table accessible by the FOCAL interpreter.

The table structure was adopted in preference to providing users with the facilities to include any selected channel address within interrupt processor mainly because of the simplicity of the chosen method when compared with the alternative. Inserting specified devices into the interrupt service routine would require the use of sophisticated handling routines and hence an increase in the amount of core used.

The function call used for accessing the data table of inputs is described later in this section.

(b) Output Counter Cards

The output counter cards are in general used for the output of a change in actuator position for control. The reasons why it was necessary to include them within the clock service routine of the interrupt processor have therefore been outlined in the section of this chapter dealing with control functions.

5.3.4.2. Directly Accessed Peripherals

As previously explained, function calls in FOCAL are arranged in tables for use with the sort and branch routines. There is therefore a limit to the number of new functions which can be included within the existing FOCAL structure without having to resort to the relocation of these tables. The limited space available for the modifications were also insufficient to allow for further table rearrangement.

It was therefore decided that instead of using separate function calls for each of the peripheral devices, a single function call would be used for input devices and another for output devices. Some timing penalty is obviously incurred by adopting this method although it was insufficient to

cause any problems.

This particular type of structure necessitates the use of an extra argument to define which of the input or output peripherals would be required.

1. Input Devices

The following function calls allow the user to access input devices:-

S	Z	=	FIN (0HRZ, CHANNEL)	Input counter cards
S	Z	=	FIN (0DPM, CHANNEL)	Digital panel meter
S	Z	=	FIN (0ADC, CHANNEL)	Analogue to digital converter
S	Z	=	FIN (0DIG, CHANNEL, BIT)	Digital inputs

Operation of the FIN function

By placing a 0 in front of the character string in the first argument, the alphabetic characters are decoded as a number. Each alphabetic character being allocated a value of from 1 to 26 according to its position in the alphabet (except for E which is used for exponentiation.)

Thus the string 0HRZ is decoded by the decimal to binary input routine in the following manner:-

$$\begin{aligned}
 H &\equiv 8 & R &\equiv 18 & Z &\equiv 26 \\
 0HRZ &\equiv 0 * 10^3 + 8 * 10^2 + 18 * 10 + 26 \\
 &= 1006_{10} \\
 &\equiv 1756_8
 \end{aligned}$$

On entry to the function subroutine, this argument has been decoded and loaded into the floating point accumulator. It is then converted into a 12 bit integer number by using a routine in the floating point package and saved. The second argument is then evaluated, converted to an integer value and saved as the required input channel address. The initial argument is then tested against a list of standard codes using a sort and branch routine. If a match is not found an error condition is signalled to the user. Finding a match causes program control to be transferred to the software driving routine for the required input device, the channel address is set and the input read into the accumulator.

As in most cases, the result of the input argument is awaited by the users FOCAL program, it would be pointless to drive these particular peripherals from an interrupt system. Also the complexity and size of skip chains in order to detect which particular device of the many available would be prohibitive.

The scaling of the input value is then accomplished in the following manner for each of the separate input devices.

Digital Panel Meters

The data input from a three digit digital panel meter produces a count of between 0 and 999 input pulses on the input counter card of channel address 0. This count is loaded into the floating point accumulator so as to set the variable assigned in the SET command to a value of between 0 and 999 on return from the function subroutine.

Analogue to Digital Conversion

The analogue to digital converter produces a 10 bit value in the accumulator for an input voltage of between 0 and 5v. The 10 bit value is loaded into the floating point accumulator in such a manner as to set the value of between 0 and 1, i.e. fraction of full scale.

Input Counter Card System

As previously described, these are serviced by the clock service routine every second. The counts from each card, stored in a data table are accessed by using the channel address argument of the function call. The floating point accumulator is loaded with the value of the count in such a manner as to set the value of the assigned variable in the SET command in terms of pulses per second.

Digital Input System

A third argument is necessary in order to specify which bit of the four bit input word is required. The four bits are read into bits 8 through 11 of the accumulator. Each of these bits is specified by designating the third argument with the codes 0A through 0D respectively.

If the required bit is set to a 0, the floating point accumulator and hence the value of the assigned variable is set to a positive value. If the bit is a 1 then a negative value results. This arrangement makes it possible to use the digital input function most easily with the IF command.

e.g., 3.03 I (FIN (0 DIG , A , 0A)) 3.05

3.04 G 3.03

3.05 C CONTINUE TO NEXT

2. Output Devices

The output function has been given the function code FOUT and implemented in the following manner:-

S Z = FOUT (0DIG , CHANNEL , BIT , STATE)	Digital output
S Z = FOUT (0ALM , CHANNEL , BIT , STATE)	Alarm output
S Z = FOUT (0DAC , CHANNEL , VALUE)	Digital to Analogue

The first and second arguments are used to define the output device and channel address required in a similar fashion to those of the FIN function. The allowed values of the other arguments are as follows:-

Digital and Alarm Outputs

Both the digital and alarm outputs require an extra argument to define which particular bit of the four bit output word is required. The definition of each bit was a third argument value of 0A through 0D (or 1 through 4) as in the case of digital inputs.

The final argument defines the required value of the state of the changeover relay and has the allowed value of 0 or 1.

Digital to Analogue Conversion

The final value of this variation on the FOUT command defines the output voltage of the digital to analogue converter. A value of between 0 and 2048₁₀ produces an output voltage of 0 to 5 volts.

In all the variations of the FOUT function, the variable used in the SET command is only a dummy variable and has no particular significance.

Output Counter Card Direct Output

A function for transmitting outputs through the output counter card system was also written. This however was given a separate function name

S Z = FINC (CHANNEL , VALUE)

Although it had been decided that the output counter card system should be clocked out (see next section on control) it was found necessary for testing purposes to be able to output directly to this subsystem.

The channel argument obviously defines the channel address highway required. The value argument specifies the output value required. As the output counter card register only has six bits, the maximum range of the VALUE argument can only be 0 to ± 63 . A full scale output will take approximately $1\frac{1}{4}$ seconds to be decremented from the counter although this does not hold up the FOCAL operation.

Again the variable assigned in the SET command is a dummy variable.

5.3.5. Control Functions

Control is achieved by linking input/output, mathematical and timing functions in some prescribed manner. This could be achieved with the FOCAL system without any further modifications as shown in Fig.23. This is an example of a programmed incremental PCI control algorithm (60), output of control parameters being achieved through the output counter card system using the foreground background programming facility.

Group 10 in the example has been written as a subroutine to be entered from an overall executive program. Changes in actuator position are computed using the PCI algorithm in line 10.14. The value obtained is summed with any previous output still remaining, the sum being restricted to ± 512 units so as to avoid integral saturation of the control loop.

Group 31 of the example is the priority task group, designed to be entered once every second as a foreground program. The actuators may be driven at their maximum slewing rate of 50 units per second by using this group in the manner shown.

The major disadvantage found in using the above method was timing, the computation of output values from the control algorithm being approximately 150 m seconds per loop minimum. The driving routine in group 31 required a further 100 m seconds per loop minimum. This would only allow a maximum of four control loops being scanned at a rate of once per second.

It was decided that these execution times were excessive, and that the only way of providing an improvement would be to write the control algorithm at assembler level as an additional FOCAL function. By adopting a fixed data table structure for storing control loop data, the time required for variable search routines could be eliminated. Also by using a clock driven output routine for the output of computed data, the time taken could be reduced further.

5.3.5.1. Lead-lag Function for Control

It was decided that if possible a digital version of a lead-lag function would be used. The reason for this was that with suitable manipulation of the various constants of such a network, various different functions could be obtained.

For example the lead-lag network whose transfer function has the form

$$G_c(S) = K \left[\frac{1 + T_m S}{1 + T_q S} \right] \quad (1)$$

where K is the static gain

T_m is the lead time constant

T_q is the lag time constant

has been used as a compensation function for Feed forward control (59) and for digital filtering purposes. Such a network can be made to approximate to a proportional plus integral feedback controller if the lag time constant T_q is made large while the lead time constant is kept relatively small.

i.e. $T_q \gg 1$

$$\begin{aligned} G_c(S) &\simeq K \left[\frac{1 + T_m S}{T_q S} \right] \\ &\simeq K \cdot \frac{T_m}{T_q} \cdot \left[1 + \frac{1}{T_m S} \right] \end{aligned}$$

which is the Transfer function of a proportional plus integral controller where:-

$K \cdot \frac{T_m}{T_q}$ is the gain factor

T_m is the integral action time

If on the other hand T_q is made very small

$$G_c(S) \simeq K \cdot \left[\frac{1 + T_m S}{1} \right] = K \cdot [T_m S + 1]$$

and therefore approximates to the transfer function of a proportional plus derivative feedback controller where:-

K is the gain factor

T_m Derivative action time

More complex control forms could also be implemented by cascading algorithm blocks together.

For computational ease equation (1) can be manipulated into a slightly different form.

$$\begin{aligned} G_c(S) &= K \left[\frac{1 + T_m S}{1 + T_q S} \right] \\ &= K \cdot \left[\frac{1 + \frac{T_m}{T_q} \cdot T_q S}{1 + T_q S} \right] \end{aligned}$$

$$= K \left[\frac{KR (1 + T_q S) + 1 - KR}{1 + T_q S} \right]$$

$$= K \left[KR - \frac{(KR - 1)}{1 + T_q S} \right]$$

Where KR is the ratio of the two time constants of the network.

The lead lag network is thereby converted from a lead and lag in series, to a lagged signal in parallel with an unlagged signal, fig.24.

The lagged signal can be generated digitally in the following manner:-

If x represents a signal and y the signal lagged by T_2 units of time:

$$x = y + T_2 \frac{dy}{dt}$$

expanding in finite difference form

$$x_n = y_n + T_2 \cdot \frac{y_{n+1} - y_n}{\Delta t}$$

$$y_{n+1} = \frac{\Delta t}{T_2} x_n + \left(1 - \frac{\Delta t}{T_2}\right) y_n \quad - (2)$$

The lagged signal part of the transfer function can be generated for the next cycle through the control loop by using the difference formula of equation (2), where Δt will represent the interval at which the particular control loop is being scanned.

This function was initially developed in a differential form for output of a change in actuator position on the output counter card system. Initial trials with this control algorithm were carried out using the equipment described in Appendix A. When using the algorithm as a P + I feedback controller on the flow control loop, large offsets were produced in response to step changes in set point. This could be attributed to many reasons, primarily that it is only an approximation to a P + I controller and does not possess the required gain and phase characteristics and secondarily rounding off errors occur when converting the floating point computed output value into an integer value for output.

The function was also very costly on core requirements, the differential form requiring eight words of core per control loop, one for the output channel address, one for storing the summed output ready for transmitting on the clock driven output counter card system, three words

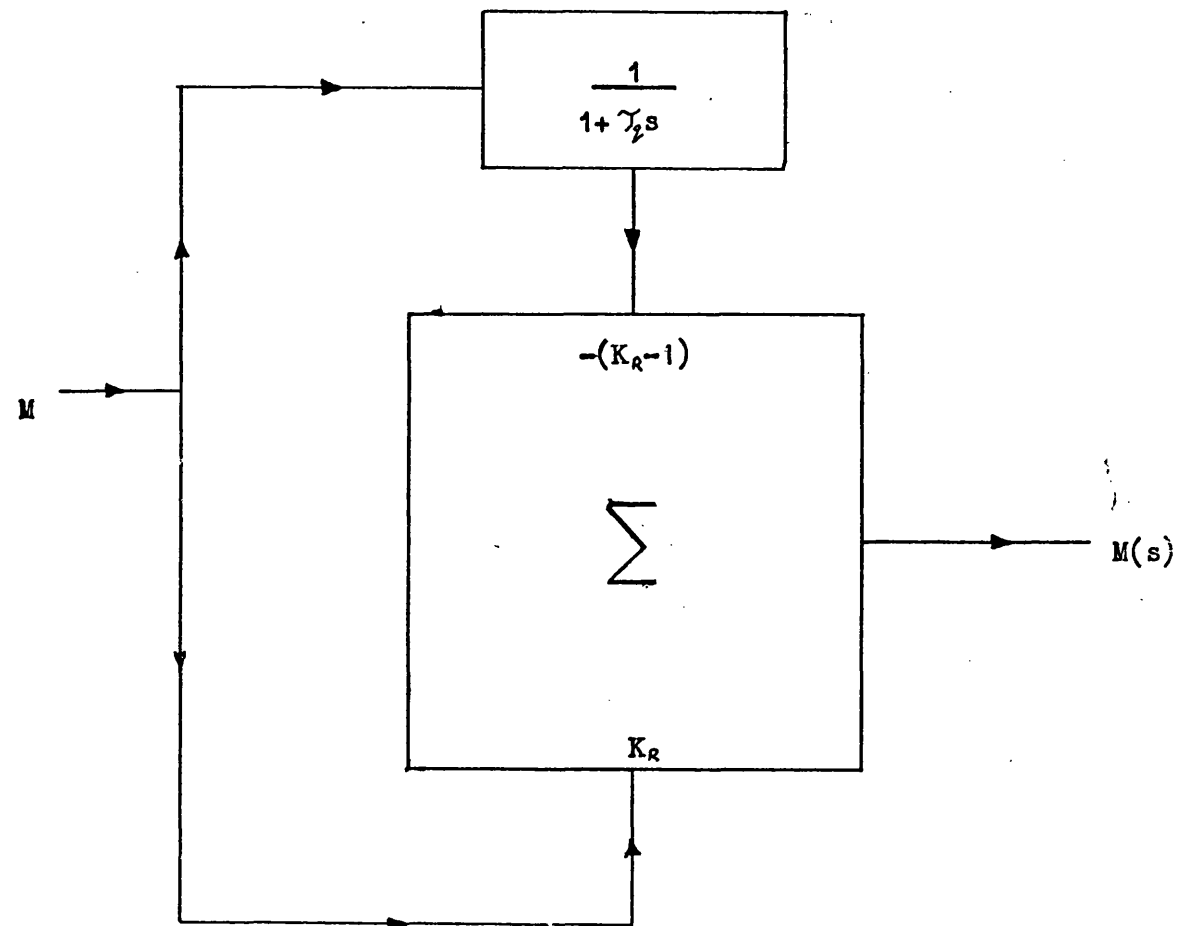


Figure 24. Simplified Lead-Lag Function

for storing the floating point value of the past measured value of the control loop and another three words to store the lagged signal value.

The function was however kept in the system, but in a non differential form. This enabled the output facility and associated manipulation routines to be omitted thereby reducing the core requirements. It was envisaged that the function could be used for feed forward compensation or for digital filtering. If necessary, it could be overwritten at some later date so as to provide space for a new user defined function.

The function of this form requires the following five parameters:-

- a. One to define the position in core of the stored lagged function value for this particular loop
- b. One for the ratio of the two time constants KR
- c. One for the lag time constant
- d. One to define the interval at which the particular loop is being scanned
- e. One for the numerical value of the function being compensated.

The loop number and function value must be provided each time the control algorithm is used, the other parameter could be changed only when necessary by inclusion of a different number of arguments in the function call. This would however require that the control parameter be stored within the data block for the specified loop requiring three words for each parameter and hence a total of twelve per loop.

The function was therefore implemented under the name of FDYN in the following form:-

$S\ Z = \text{FDYN} (A, B, C, D, E)$

where A = loop number to define the data block

B = The function value to be compensated

C = Lag time constant

D = Scan time constant

E = Ratio of time constants

The first argument is used to set up the data block in which the lagged signal value is to be stored. The rest of the arguments are picked up and manipulated using calls to the floating point interpreter so as to produce the compensated value of the input function and also to compute the lagged signal for the next entry of the loop. The floating point

accumulator is finally loaded with the compensated function value and return is made via the normal function return in FOCAL so as to set Z to the value of the compensated function.

A facility has also been included for initialization purposes when the data area will need to be cleared. This takes the form

$$S Z = FDYN (0CL)$$

Z this time is used as a dummy variable, the 0CL being detected in the initial section of the routine and the data area completely cleared.

The function call in full form takes about 35 m seconds to be completed which is a vast improvement on the sorts of times one could expect for a similar algorithm written in FOCAL; the penalty paid is that of core. Three words are required per loop, and a maximum of 6 loops have been allowed for.

5.3.5.2. PCI Control Algorithm

As the initial efforts of providing a multi purpose control function were unsatisfactory, it was decided that a more specific type of control algorithm would be required. Experience gained with the use of a PCI (60) control algorithm when writing the algorithm in FOCAL showed that it was simple to use and provided good responses.

For a normal P + I controller, an equation of the form:-

$$P = K_c \left[e + \frac{1}{T_i} \int e dt \right] \quad - (3)$$

is used.

Where P is the position of the actuator driven by the controller

K_c is the gain factor

e is the error value between the set point and the measured value

T_i is the integral action time constant

As the output from the control algorithm is to represent the change in actuator position for output on the output counter card system, equation (1) must be expressed in differential form.

$$\frac{dp}{dt} = K_c \left[\frac{de}{dt} + \frac{e}{T_i} \right]$$

and in difference form

$$\Delta P_n = K_c \left[\Delta e_n + \frac{\Delta t}{T_i} e_n \right] \quad - (4)$$

where ΔP_n is the change in actuation position at the n^{th} sampling interval

e_n is the error at the n^{th} sampling interval

Δe_n is the difference between the error at the n^{th} and $n - 1^{\text{th}}$ sampling intervals

$$e_n = S_{pn} - MV_n$$

S_{pn} is the set point

MV_n is the measured value at the n^{th} sampling interval

$$\begin{aligned} \Delta e_n &= S_p - MV_n - S_p + MV_{n-1} \\ &= -(MV_n - MV_{n-1}) \end{aligned}$$

If the set points remain constant

Equation 4 can be expressed in the following form

$$\Delta P_n = K_c \left[-(MV_n - MV_{n-1}) + \frac{\Delta t}{T_i} (S_p - MV_n) \right] \quad - (5)$$

This algorithm is the same as a P + I controller if no change in set point occurs, however it is reputed (60) to provide better response to set point changes than is a P+ I algorithm.

For the same reasons as explained in the development of FDYN all of the parameters have to be transferred each time a control loop is entered, sacrificing time for core utilisation. The function call is therefore of the form

$$S \ Z = FCON (LOOP, MV, SP, IT, SC, KG, CHANAD)$$

where LOOP is a loop number to define a data block

MV is the measured value for the control loop

SP is the set point

IT is the integral action time

SC is the interval at which the loop is being scanned

KG is the gain factor

CHANAD is the output channel address

Z will be set to the computed output value

For effective direct digital control, the maximum slewing rate of the actuator must be maintained (61); This means that it is essential for calculated control outputs to be driven out through the output counter system by using the clock routine in the interrupt processor. The output counter cards have a maximum slewing rate of 50 units per second and a

maximum actuator travel of 512 units.

The interrupt processor was therefore extended in order to service the output counter cards with the calculated control outputs. The table of outputs is examined every second; if no output is present then the routine skips on to the next in the table until all has been examined. If an output value is found, its magnitude and sign are tested. If the value is greater than 50 then the maximum output of 50 is driven out on the required channel with the correct strobe (i.e. + and -) and the remainder is restored. Output values of less than 50 units are merely driven out and the store cleared.

For speed of operation in the interrupt processor, it is essential that the arithmetic operations are done in single word integer arithmetic. As the floating point package is essentially non re-entrant, it cannot be entered from the interrupt processor. Conversion of the output calculated in floating point must therefore take place within the control routine.

The floating point output is therefore converted into a 12 bit integer number, tests being made to see that it does not exceed ± 2046 , added to any previous output still waiting to be clocked out. The resultant integer is then tested for magnitude. If it exceeds the maximum travel of the actuator in either direction, the resultant is replaced by an integer value equivalent to the maximum travel and stored in the output table. Integers of less than maximum travel are stored immediately in the output table.

In order to provide the control function with an extra degree of flexibility, the channel address argument has been included in the function call. Channel addresses could have been incorporated within the loop number, allowing output only on fixed output channels of 0 to 6. This could however require swapping users data lines to the computer, each time a different experiment or research programme is implemented on the computer. The channel address is therefore stored within the data block for the particular loop so as to define which output address highway is required by which loop.

The channel address argument has also been made to act as a switch. When initialising a particular control loop, the value stored as the past measured value could be set to any arbitrary value. Using it as a bona fide

past measured value could result in a spurious output being generated. It is essential therefore to be able to load the past measured value store without sending an output. This facility will also be useful if cascading control loops, or debugging a program. If the 7th argument in the function call, representing the channel address, is included, the output table is filled as described and the calculated change value is returned to FOCAL in the floating point accumulator in the normal fashion. If however the 7th argument is omitted in the function call, the output table is not filled, the change value being returned to FOCAL only.

In this form the storage requirements of the function are five words per block, one for channel address, one for the stored integer output value and three words to store the floating point value of the past measured value of the controlled variable.

The facility for totally clearing the data blocks has also been included, the available forms of the function call are therefore

```
S Z = FCON (0CL)
S Z = FCON (A, B, C, D, E, F)
S Z = FCON (A, B, C, D, E, F, G).
```

The first argument is either used as a switch for the clear instruction or as a loop number. The rest of the arguments are then picked up and loaded into the floating point accumulator by using the argument evaluation routine, arithmetic manipulation being carried out by calls to the floating point interpreter. The channel address argument is searched for and if found, the floating point value is converted into a 12 bit integer value and stored in the manner previously described. After output sorting or immediately in the case of no channel address argument, the floating point accumulator is loaded with the computed output value. This ensures that the variable named in the SET command, when calling the function, is given the increment value when control is returned through the normal function return.

5.3.5.3. Timing and Core Allocation of the Control Algorithm

The completion of this particular control algorithm takes approximately 50 m seconds per loop which is about one-fifth of the time taken for the same algorithm if written in FOCAL language. Admittedly some time is also used for

data manipulation in the interrupt processor but the total time taken in the interrupt processor is less than 500 μ seconds.

A limit had to be imposed upon the number of control loops available as five words of core were required for storage purposes in each loop. Seven loops have been allowed, requiring a loop number of between 0 and 6 but no restrictions have been placed upon the channel address to be used for the control loops.

5.3.6. Communication With Operating or Program

An essential feature of a real-time operating system is the ability to modify the value of a parameter within a program while the program is running, for example, the manual control of the set point of a control loop, or for setting a flag to initiate the start up or shut down of an item of equipment. This process of communication should not hold up the execution of the program except for the brief period of time necessary to reset the value of the parameter.

If special hardware had been available such as an operators control panel, this process could have been included by use of a service routine within the interrupt processor. However with only a teletype keyboard available as a communication device it was necessary to make adaptations to the FOCAL structure so as to allow communication via the teletype while the program is still operating.

The facility has been incorporated into FOCAL by again making use of the end of line break point, allowing the internal character handling routines of the FOCAL interpreter to be used.

It has been arranged that at the end of every line of the FOCAL program, the teletype input buffer is examined for input. Once the character handling routine has been activated, by typing a CTRL/S character (ASCII code 223) at the terminal keyboard, subsequent characters, when detected, are packed into the command input buffer. Upon receipt of a carriage return character a direct SET command is forced, operating on the text which has been packed into the command input buffer. The routine is then de-activated and the program resumed. Activation and completion of the routine are signified on the teletype device by the characters > and < respectively.

In order to provide a suitable hard copy of the parameter modified it was found necessary to inhibit any TYPE statement while the parameter modification routine was activated. The reason for this being that output characters are entered into a first in, first out ring buffer for echoing.

It was also found necessary to inhibit any ASK statements so as to avoid corruption of input data.

As both TYPE and ASK commands use common system software routines both could be inhibited by including a JMP instruction at the beginning of the TYPE/ASK subroutine.

This protection system is enabled immediately the CTRL/S character has been given and is only cleared when the parameter modification has been completed.

Thus if a TYPE or ASK statement is encountered during parameter modification, the program is halted until the modification operation has been completed. The TYPE or ASK statement is then re-activated and the program resumes from the position where the program was halted.

The handling routine makes use of the character reading and packing routines available within the interpreter which are re-entrant at the break point chosen.

Since many flags and inhibit switches are set during the parameter modification it was necessary to extend the error recovery routine to clear the flags etc. in the event of an error occurring during the process of modification.

Fig.25 illustrates the use of the parameter modification routine, the underlined character being echoed by the computer.

5.3.7. Fault Protection

Limited fault protection has been built into the system by including standard routines for handling the power failure protection option as described in chapter 3. Also within the clock routine of the interrupt processor, the output counter card system is protected by updating the watchdog timer once every second.

5.3.8. Error Detection

In most of the functions developed for real-time FOCAL, a certain degree of error protection was necessary so as to avoid the possibility

```

C-8K REAL TIME FOCAL @1974 WITH DYN

01.01 A "TIME PLEASE",!, "SECONDS",S,I, "MINUTES",M,!
01.02 A "HOURS ",H,I, "DAY ",D,I
01.03 S Z=FTIM(OST,S,M,H,D);D 3
01.04 D 2
01.05 I (FTIM(OSCS))1.04,1.04
01.07 I (TI)1.09
01.08 G 1.05
01.09 S TI=0;G 1.04

02.01 S S=FTIM(OSCS);S M=FTIM(OMNS);S H=FTIM(OHRS)
02.02 S C=S+100*M+10000*H;T %06.0,C,!
02.03 F I=1,500;S V=I
02.04 C LAST LINE IS A TIME DELAY

03.01 T " H M S",!
*G
TIME PLEASE
SECONDS:00
MINUTES:34
HOURS :09
DAY :18
H M S
= 93400
= 93500
>TIME=-1
<
= 93519
>TIME=-1
<
= 93532
= 93600

```

User depresses CTRL/S keys and the computer echoes the underlined character. The remainder of the line is entered by the user and terminated with a carriage return. The computer then echoes the second underlined character to signify that the command has been accepted.

Figure 25. Example of the use of the On-line Parameter Modification Facilities

of users corrupting the system. For this purpose, the error recovery routine available in FOCAL was used extensively. A list of error codes and their purpose is given in Fig.26.

5.3.9. Timing of the Functions

Fig. 27 gives a list of all the new functions developed with their respective execution times.

5.4. Conclusions

In adding all the required enhancements to FOCAL, it became obvious that more core space than there was readily available would be required. Thus it was necessary to overwrite some of the existing "extended functions" present in FOCAL.

The initial version of Real Time FOCAL, REAL TIME FOCAL : DYN included all of the enhancements described within this chapter however the standard FOCAL function FATN, FEXP, and FLOG had to be excluded. The core configuration is as shown in Fig.28.

At some later stage it was found necessary to exchange the FDYN function with a standard FLOG function for a final year undergraduate project on the examination of the dynamics of a heat exchanger (62). Fig. 29 shows the core configuration of this second version REAL TIME FOCAL : LOG.

As a test of the FOCAL system experiments were carried out on a Heat Exchanger system, the results of the test have been included in Appendix A.

It was found that the system worked perfectly although a restraint was obviously imposed by the interpretive nature of the language. This however is not too a severe restraint:-

- (a) Because most of the data rates dealt with in the laboratory are slow.
- (b) If a high data rate application is found, then a new function for FOCAL could easily be written. This would allow initialisation to take place from the high level language and result in the execution of a dedicated subroutine followed by eventual return to the FOCAL interpreter.

Subsequent experience gained in the use of the system, both for undergraduate and postgraduate (63) work has not highlighted any major deficiencies in the system. The main advantage of the system is its ease of use, requiring only a few hours use for any potential user to be able to use the system to its full capacity.

The ability to operate the system from a remote terminal also makes its use ideal in the laboratory environment where the operations terminal can be located near his experiment and not necessarily in the neighbourhood of the computer.

<u>Error Code</u>	<u>Diagnostic</u>
13.44	Normal exit from LIBRARY command
13.;6	Illegal function call in FIN or FOUT functions
15.:6	Missing argument in function call
15.<5	Too many arguments in FLAG function
16.04	Illegal character in multiple argument handler
16.59	No scan flag of that value has been set up
16.69	Illegal code in FTIM function
18.03	Too many arguments in FIN, FOUT or FINC function calls
18.08	First argument of FCON function call should be positive
18.17	Control loop number outside range(0-6) in FCON function call
18.61	Too many arguments in FCON or FDYN function call
18.;3	Missing argument in FCON function call
19.72	Missing argument in FINC function call
20.22	Negative argument in FLOG function call
20.51	First argument of FDYN function call should be positive
20.60	Control loop number outside range in FDYN function call
20.97	Too many arguments in FDYN function call
21.<4	Illegal character or line not present in MODIFY command
31.05	Terminator other than or , or ; or RETURN used in variable erase command E F,
31.46	Character other than A,C or D used in ERASE command and extensions

Figure 26. Error Diagnostics for Real-Time FOCAL

<u>Function Call</u>	<u>Arguments</u>	<u>Action</u>	<u>Execution time, msec</u>
S Z=FTIM(OST,A,B,C,D)	A=seconds value B= minutes value C= hours value D=days value	Initialises clock counters	22
S Z=FTIM(X)	X=QDYS, OHRS, OMNS, OSCS	Z is set to value of clock counter designated by X	13
S Z=FLAG(OST,N,A,B,C)	N=clock interrupts per second A,B,C=scan intervals in clock pulses	Initialises scan flags	35
S Z=FLAG(A)	A=one of the three scan flags	Sets Z negative on the appearance of designated scan flag	14
S Z=FIN(OHRZ,A)	A= channel address	Sets Z to the value of the count in input counter card of channel A	19
S Z=FIN(OADC,A)	A= channel address	Performs A to D conversion on channel address A Z= 0 to 1 for input of 0 to 5v	17

Figure 27. Real-Time FOCAL Functions and Approximate Execution Times

<u>Function Call</u>	<u>Arguments</u>	<u>Action</u>	<u>Execution time, msecs</u>
S Z=FIN(ODPM,A)	A= Required Panel meter	Sets Z to the value supplied by a 3 digit panel meter, 0 to 999	40 to 80
S Z=FIN(ODIG,A,B)	A= group address B=bit code	Senses contact status Z is negative if contact is open	25
S Z=FOUT(ODAC,A,B)	A= channel address B= output value 0 to 2048	Performs a D to A conversion output in range 0 to 5v	26
S Z=FOUT(ODIG,A,B,C)	A= group address B= bit code C= desired status	Changes contact status	27
S Z=FOUT(OALM,A,B,C)	A,B,C as for digital output	Setting C to 0 or 1 activates or clears selected alarm	27
S Z=FCON(A,B,C,D,E,F,G)	A=loop number B=measured value C=set point D,E,F=controller settings G=channel address	Provides feedback control using incremental PCI algorithm	42

Figure 27. Continued

<u>Function Call</u>	<u>Arguments</u>	<u>Action</u>	<u>Execution time, msec</u>
S Z=FDYN(A, B, C, D, E).	A=loop number B=variable to be compensated C, D, E=algorithm parameters	Provides dynamic compensation for use in feed-forward control Z is set to the compensated value of B	35
S Z=FUNC(A, B)	A= channel address B=output value 0 to 63	Operation of output counter card system independently of the clock	14

Figure 27. Continued

Real Time Focal : Dm Core map Field 1	
0	Zero Page Pointers
1	Command Decoder
2	Getln
3	Control & Transfer
4	Control & Transfer
5	Control & Transfer
6	Control & Transfer
7	Control & Transfer
8	Control & Transfer
9	Control & Transfer
10	Control & Transfer
11	Control & Transfer
12	Control & Transfer
13	Control & Transfer
14	Control & Transfer
15	Control & Transfer
16	Control & Transfer
17	Control & Transfer
18	Control & Transfer
19	Control & Transfer
20	Control & Transfer
21	Control & Transfer
22	Control & Transfer
23	Control & Transfer
24	Control & Transfer
25	Control & Transfer
26	Control & Transfer
27	Control & Transfer
28	Control & Transfer
29	Control & Transfer
30	Control & Transfer
31	Control & Transfer
32	Control & Transfer
33	Control & Transfer
34	Control & Transfer
35	Control & Transfer
36	Control & Transfer
37	Control & Transfer

Real Time Focal : Dm Core map Field 1	
0	Command Input Buffer
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	TEXT
11	
12	
13	
14	
15	
16	
17	
18	
19	
20	
21	
22	
23	
24	
25	
26	
27	VARIABLES
28	
29	
30	
31	
32	
33	
34	
35	
36	PUSHDOWN LIST
37	Loaders

Figure 28.

Real Time Focal : Log Core map Field 0

0	Zero Page Pointers											
1	Command Decoder					Getln						
2	Fntabf		Do			Pushdown List Controls						
3	Control & Transfer		Write			Teste		Sorte		Grptst	Input	
4	Comlst		If		Set & For				Dys & Int		Conpro	
5	Type & Ask				Modify			Sortj		Adc, Outl and others		
6	Getarg							Spnpr, Testn, Ran, Popj and others				
7	Ecall											
10	Terms Egn, Abs et al		Tstlpr, Partest			Delete				Reade		Fntabl
11	Erase			Findln			Getc				Endln	
12	Hsp, Prmtln, Prnt and Printc					Puckc					Extensions	
13	Interrupt Processor, I/O Routines and Error Recovery Routine											
14	Rubout			Symbol Table Dump				C/P buffer		Gn12, Gn13, Gn14		
15	Extensions to Library and Variable Search					Echo Disable		Dirchk, Sortnset, Getadd				
16	Parameter Modification Routine											
17	Clock Service Routine Extension to the Interrupt Processor									Flag continued		
20	Getarg(*)		Power Failure Routines			Flag		Pfin		Lists		
21	Fin and Fout Commands											
22	PCI Control Algorithm											
23	Synchronous Service Routines					Fine		PCI Algorithm Data Table				
24	Pran		Log									
25	Cos and Sin									Mod and Alt ode		
26	FLOATING POINT											
27	INPUT AND OUTPUT											
30	ROUTINES											
31						High Speed Reader Routine						
32												
33	FLOATING POINT INTERPRETER											
34												
35												
36	Sgt					Library						
37	Variable Erase Routine			E C, E D		Further Extensions - Run Switch						

Real Time Focal : Log Core map Field 1

0	Command Input Buffer									
1										
2										
3										
4										
5										
6										
7										
10	TEXT									
11										
12										
13										
14										
15										
16										
17										
20										
21										
22										
23										
24										
25										
26										
27	VARIABLES									
30										
31										
32										
33										
34										
35										
36	EPIGRAPH LIST									
37	Loaders									

Figure 29.

CHAPTER 6.

A Time Shared Real-Time FOCAL System

6.1. Time Sharing the FOCAL Interpreter

As discussed previously, the primitive interrupt structure and the absence of any direct memory access mass storage device had eliminated the possibility of developing an efficient multi-user time shared operating system for the PDP-8 based upon the use of a high level language. It was however thought to be possible to provide a minimal time shared system by extending the capabilities of the FOCAL interpreter.

The FOCAL interpreter operates by scanning the text of a program, interpreting and executing the statements of the program. Parameters regarding the state of the program are constantly maintained, so that at any particular time, a record is available as to the exact state of execution of the program. Thus it is possible to terminate the execution of one program, replace the program and associated parameters with a different program and its associated parameters and then continue the execution of the new program.

This type of process would require the inclusion of some form of Executive Routine within the interpreter structure so as to determine when program exchanges could occur and hence share the interpreter between two or more programs. If facilities were provided whereby the programs were created by different users, then the system would essentially be sharing its processing time between different system users.

As the PDP-8 system does not possess any external mass storage device, all programs would have to be stored within the area currently available for program storage in the single user version. This would result in a reduction of the space available for individual programs and hence the maximum size of program which could be accommodated within the time shared system.

Another disadvantage of such a system would be that of the reduced execution speeds for a single program which would be incurred because the available processing time would be shared between a number of programs.

Experience gained when using the single user version of Real Time FOCAL showed that in general, programs for on-line use would not necessarily be large. Also the execution speeds achieved were adequate for the type of work which was envisaged.

Thus it was decided that the FOCAL interpreter could be enhanced in order to provide on-line computing facilities for more than one user on a single core only computer.

6.2. Design Constraints

The prime constraints of a shared system are:-

- a. A user should not be able to access or corrupt any part of any other users program. This essentially means that each user must be allocated a fixed amount of core for program storage and then provided with protection facilities so as to ensure that no other user can access that area.
- b. The possibility of a user waiting for a I/O device to become available and thereby temporarily holding up the operation of another users program should be prevented
- c. Available processing time should be divided in such a way as to provide each of the system users with virtually equal shares.
- d. Sufficient space must be made available for adding real-time functions to the FOCAL system and if at all possible, the facilities made available in the single user version of Real-Time FOCAL should be made available in the shared version of FOCAL.
- e. The processing time used for exchanging users and servicing console devices should be minimised.

6.3. User Exchange Techniques

The parameters specifying the current state of a particular FOCAL program are located within the first half of zero page in the same memory field as the interpreter. At program/user exchange time, the parameters of the new program must be inserted into their respective locations in page zero, so as to control the operation of the new program. At the same time, the parameters of the old program must be saved until such time as the operation of that program can be recommenced. In order to provide each of the system users with equal shares of the available processing time, this process of data interchange should really be clocked

in some manner and users should only be allowed to remain in an operational state if no other user is waiting to use the computational facilities at a specified exchange time.

Unfortunately the internal subroutines of FOCAL are non re-entrant. This would involve the saving of all subroutine return addresses and data stores used within the subroutines, for each of the system users, if an external event was used to trigger the user interchange process. This would require large amounts of core for storage purposes and also an increase in the time required to exchange users. It would therefore be more convenient if some other method could be used for initiating the exchange of users.

This can be achieved in FOCAL by making use of the fact that either at the end of every line or the start of every sub line, no subroutine is in the position of being partially executed. At this point, any of the subroutines could therefore be used without having to save return addresses or data stores.

The above techniques would not provide exactly equal shares of the processing time for each of the system users. The inequality could however be minimised by the selection of the "break point" which occurs most frequently, i.e. the start of each sub line. The savings in swapping times and core used by such a technique would outweigh the inequalities in time allocations involved particularly when dealing with a core only mini computer.

6.4. Terminal Input/Output Handling

If a users program was allowed to wait, either for input from its associated terminal device, or for output of a character to that terminal device when the terminal device is still in the process of handling the last character, a lot of valuable processing time could be wasted. The possibility of a single user holding up all other users in this situation could be avoided if all input/output transfers between the users program and its associated terminal device were buffered.

Thus if any users program requests input from its associated terminal device, the input buffer must first be examined to determine whether a character is available. If an input character was present in the

buffer, the program could continue in the normal fashion. Alternatively, if no input was available, it would be essential to put this particular user's program into a wait status and select another user's program. When input data eventually becomes available in the input buffer, the wait status of the program could be cleared and the program swapped into the active status in due course.

Similarly with character output. If the terminal output buffer becomes full and the program is waiting for another free space in the buffer for the output of further characters, the program should again be put into a wait status. The interrupt processor could be used to fill the input buffer and empty the output buffer thereby making the operation of the terminal device virtually independent of the state of execution of the program.

This particular technique, although allowing effective use of the available processing time, necessitates the exchange of users at a position where a subroutine has been partially executed. It would therefore require the saving of all subroutine return addresses associated with terminal I/O transfers and all possible active stores used within those routines in addition to the parameter set located in zero page.

6.5. Core Allocations and Design Criteria

The core only configuration of the PDP-8 computer system necessitated that a limit be imposed upon the maximum number of users which could be accommodated by the system. This limit was determined by the following factors:-

- a) The amount of core available after the real-time extensions, user exchange Executive routines and terminal handling routines have been integrated into the FOCAL interpreter.
- b) The minimum acceptable area of core required for a users program.

Experience, which had been gained in developing FOCAL programs for data acquisition and control purposes, had shown that the minimum acceptable core space for a program * would be about the same as that provided in the 4K version of FOCAL (Chapter 3, Figure 6). This magnitude of core area for users program * storage could have been achieved by an adaption of the 4K FOCAL system in the following manner:-

In the 4K system, the users program * is accommodated in the same field as the interpreter. The user exchange executive routines of a multi-user system could be used to exchange this core area, holding an active program * with another core area holding a non-active program *. The executive routines, terminal handling routines and non-active users programs * and associated running parameters could then be accommodated within the unused 4K of an 8K computer system.

Such a system has been adopted for the four user off-line version of FOCAL, QUAD (47), which operates with the core configuration illustrated in Fig.30. Three non-active user buffers are made available in field 0, together with the Executive Routines and terminal handling routines. The active user program area is located in field 1 with the interpreter. This core configuration also provides a limited amount of free space for extensions to the interpreter.

This particular configuration has been used as a basis for a real-time multi-user version of FOCAL (49) but with only limited capabilities.

This technique of time sharing employed within the QUAD system has numerous disadvantages, amongst these are:-

1. Swapping the whole of a users program in addition to the zero page pointer and I/O subroutine parameter entails that approximately 1K of core must be transferred in both directions, i.e. active to non-active and non-active to active. This takes approximately 55 m seconds in the executive routines of QUAD, during which time approximately six FOCAL commands could have been executed. Obviously some time will be regained, as less of the available processing time will be wasted by a single user becoming I/O bound. The nett result is that approximately one-third of the available processing time is used by the swapping routine even when users are exchanged once every four sub-lines. The remainder of the time is divided between system users. Since an

* program refers to text, associated variables and pushdown list.

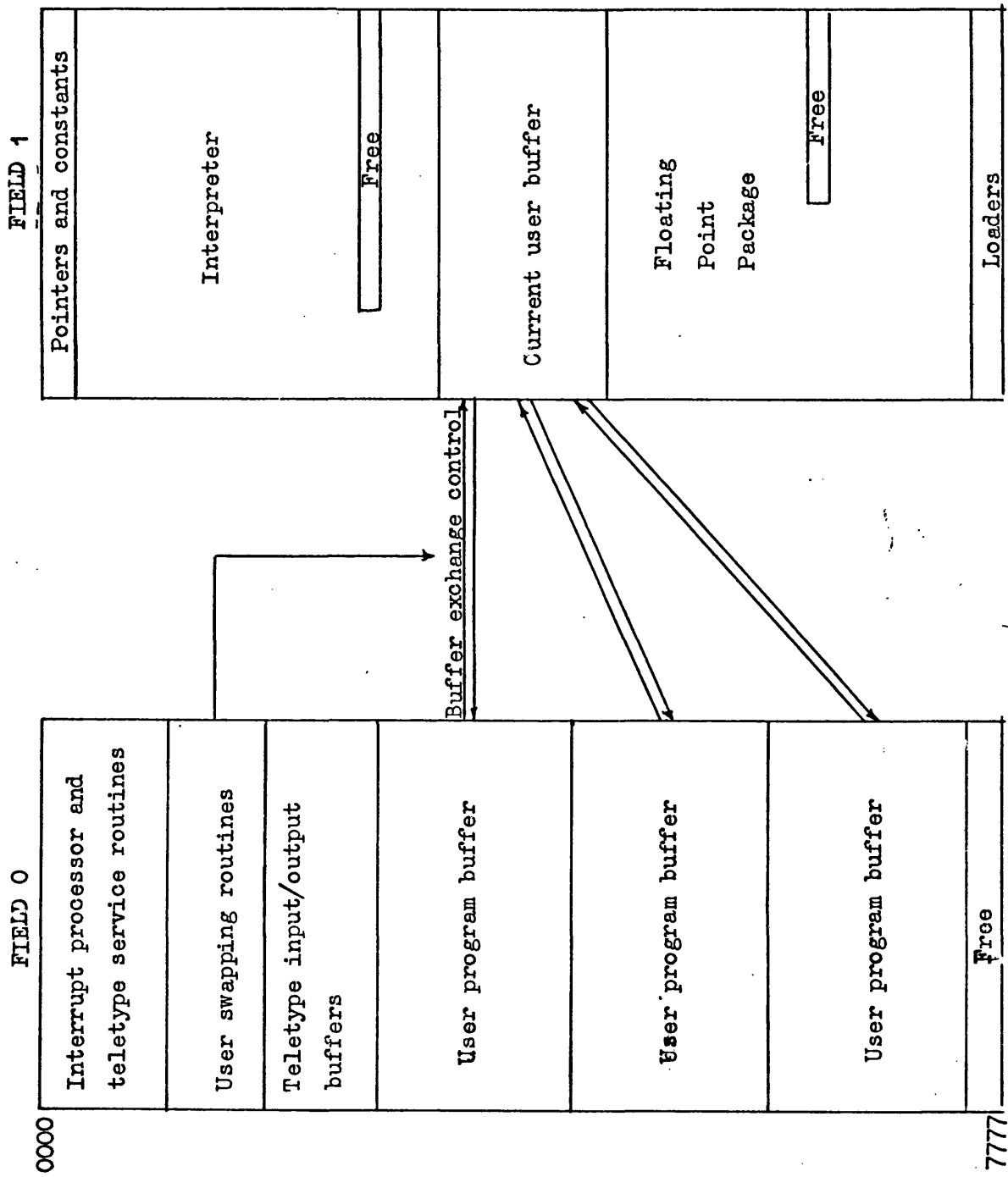


Figure 30. QUAD core configuration

interpretive mode of program execution is inherently slow, an additional time loss of this magnitude would severely reduce the response of the system.

2. In view of the core required for extensions to FOCAL for single user real-time application, it would be preferable if a little more spare core space was available than that present in QUAD. Also for the same reasons as were described when developing real-time FOCAL, it would be preferable if the available core were located in the same field as the interpreter.
3. Although the user space available within QUAD is the same as in 4K FOCAL, this is the absolute minimum amount which could be tolerated and it would be an advantage if it could be extended.

An allied problem to this is the use of the dynamic storage allocation techniques of 4K FOCAL. As described in Chapter 4, this is an annoying feature as variables are automatically erased every time the program is modified.

All of these disadvantages could be easily overcome by adopting a different method of user core allocation. If each of the system users was allocated a fixed area of core within the field not occupied by the interpreter, only the zero page pointers, I/O subroutine return addresses, active register and the pointers defining the program area would need to be saved at user swapping time. This would mean that the transfer of about 120 words of data would be involved, requiring approximately 6 m seconds, an essential improvement upon the 55 m seconds taken by QUAD.

The core area used within QUAD for the active program would be released completely allowing sufficient space within the interpreter field to include all facilities available within Real-Time FOCAL.

The available core for program storage could provide three users with about the same amount of program space as was available in QUAD. This as previously discussed is not really sufficient and it would be preferable if the available space was divided between only two users and not necessarily in equal proportions.

It was therefore decided that a two user version of FOCAL should be implemented. The envisaged core configuration is as shown in Fig.31. It is in many ways similar to the core configuration adopted in Real-Time FOCAL with the program storage area located in a different field to the interpreter. Many of the facilities added to Real-Time FOCAL could therefore be implemented providing a two user system compatible with the single user system.

The system executive would control the execution of users programs on a Roll-in Roll-out basis using the start of line breakpoint as a swapping point. So as to reduce the amount of time spent exchanging users, swapping will essentially be carried out every four sub lines, providing facilities for altering this value should the response prove inadequate. A user in an I/O wait status awaiting the availability of his terminal device would be inhibited from becoming active, thereby avoiding the possibility of an I/O bound user holding the system up.

Input/Output character transfers would be buffered, the terminal device being interrupt driven.

6.6. Executive Routines of the Time Shared System

The executive routines which allow two users to execute programs and effectively time share the interpreter are similar in structure to those employed in QUAD. They can be subdivided into the Interrupt Processor, Teletype Input and Output Service Routines and User Swapping Routines.

As a program interrupt causes an automatic subroutine jump call to location 0000 in field 0 it would be convenient if the interrupt processor and teletype service routine were located in field 0. The Executive Routines were therefore developed to reside in field 0 and operate upon the resident interpreter in field 1.

6.6.1. The Interrupt Processor

This operates using a skip chain technique. It has been arranged to service interrupts from the following devices:-

- a) Power failure protection option. Detection of Power low condition causing the active register of the computer

FIELD 1

FIELD 0

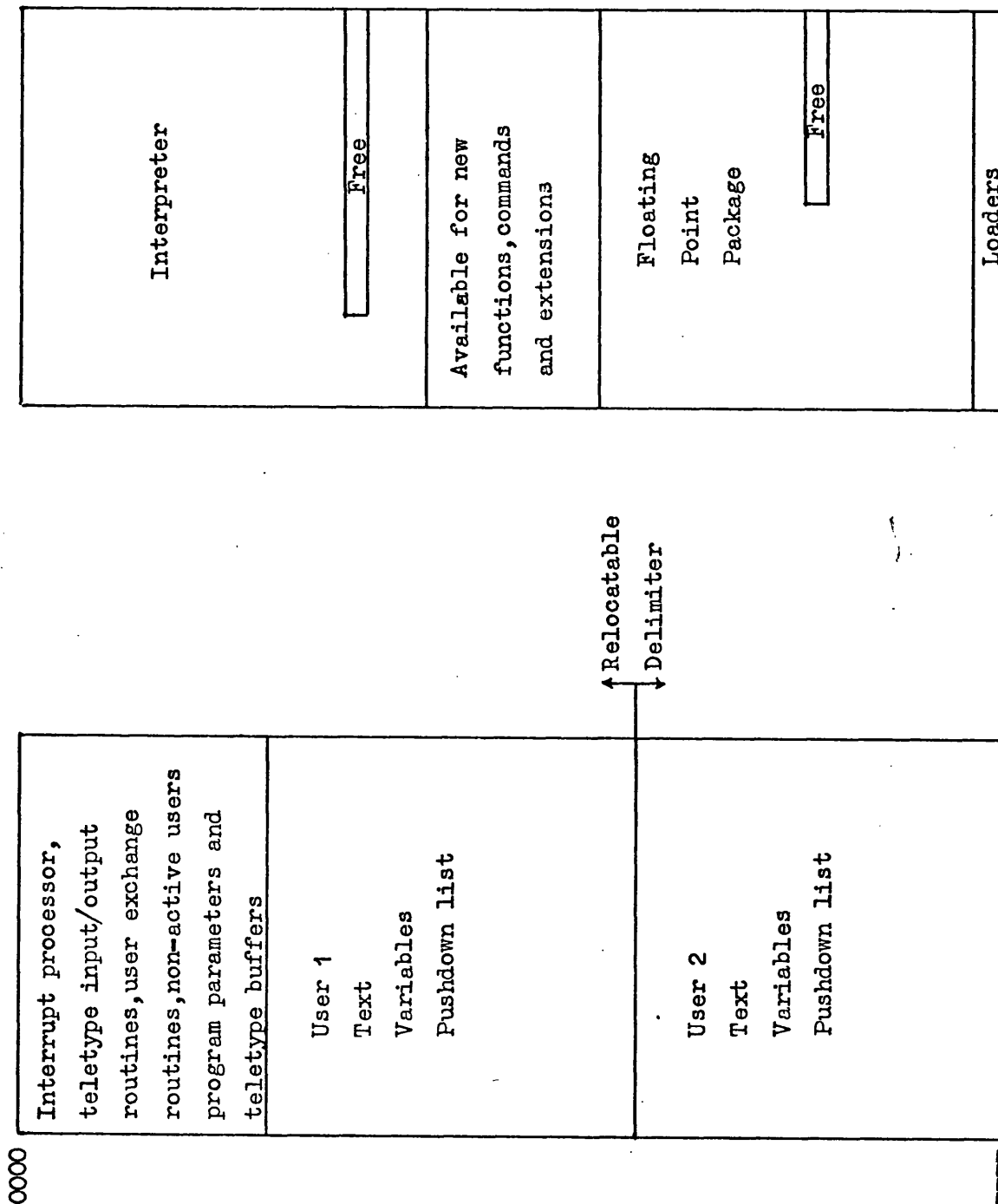


Figure 31. Proposed Core Configuration for A Two User Real-Time System

to be saved ready for automatic restart once the power level returns to the normal state.

- b) The teletype terminal devices of both system users. The detection of an interrupt from an input terminal device causes the keyboard service routine to be entered. The detection of an interrupt for an output terminal device to signify that it is ready to receive another output character causes the **printer** service routine to be entered.
- c) The real-time clock. (See Section 6.9 on timing considerations) Every clock interrupt causes a single set of counters recording current time and the counters used for users scan flags to be updated. If any of the scan flag counters "rolls over" to zero, the scan flag associated with that particular counter is set to a negative value and the counter value re-initialised. As each user has been allocated three scan flags, a total of six counters must be updated at each clock interrupt. Each second the watchdog timer on the output counter card protection system is updated; the input counter cards on channel addresses 1 to 6 are read and reset, the count values being stored in an input table. Provision has been made to service outputs for the output counter cards on channel addresses 0 to 9 if there is any output data available in the output table.

6.6.2. User Status Record Manipulation

As only one user can be in an active state at any one time, it is necessary to hold information as to the state of execution of a non-active program. Such information would include the restart address for when the program becomes active again, the next available position in the teletype output buffer, the position in the input buffer which should contain the next input character. Information would also be required as to whether a particular user is an I/O wait status or similar condition.

Similarly when dealing with interrupts from terminal devices, it is necessary to know which users terminal has caused the interrupt, where in the output buffer the next character for output can be found, where in the input buffer the next input character should be stored.

In order to maintain a record of all this information, each user is provided with a set of parameters, Fig.32, which are normally located in a "base" area of core. In order to update this set of parameters it is necessary to move them from the base area of core into the active locations in zero page (Fig.32) and upon completion of all the updates, the parameters are returned to the base area of core. The routines which perform these functions are ONDECK and OFFDECK respectively.

The occasions at which updating of the pointer is necessary are:-

1. At user swapping time where a record of the state of the new non-active users program must be maintained.
2. Either when a character is entered into the output buffer or requested from the input buffer by a FOCAL program, the buffer control pointers must be updated.
3. In the event of an interrupt from either a keyboard device a character will be entered into an input buffer, requiring that the input buffer input pointer be reset.
4. In the event of an interrupt from a printer device a character will be extracted from an output buffer, if available, and the output buffer output pointer must be reset.

6.6.3. Keyboard Service Routine

This routine has been designed to service an interrupt from either of the keyboard devices in the following manner:-

The skip chain in the interrupt processor maintains a record of which user's device interrupted the program. The routine ONDECK is then used to transfer this user's parameter set into page zero.

The IOT code held within the parameters is used to form a read instruction, and the character is read in to the processor accumulator. The input character is first tested for a control code which performs various functions within the system., e.g.:-

<u>Character</u>	<u>ASCII code</u>	<u>Function</u>
CTRL/C	203	Program abort
CTRL/R	222	Teletype Echo Disable
CTRL/T	224	Teletype Echo Enable
CTRL/S	223	Online parameter modified

<u>Location</u>	<u>Tag</u>	<u>Function</u>
0105	PCM	Active users program counter
0106	OBUFFO	Teletype output buffer output pointer
0107	OBUFFI	Teletype output buffer input pointer
0110	OBUFFO	Reset value for active users input and output teletype buffers
0111	IBUFFI	Teletype input buffer input pointer
0112	IBUFFO	Teletype input buffer output pointer
0113	DECKP	Active users status word. See figure 34 for detailed explanation
0114	IOTX	IOT code for active users teletype

Figure 32. Active Users Status Parameters

Ordinary input characters are echoed at the user's console via the routine ACTION Q, provided that the echo facility has not been disabled. The character is then loaded into the character input buffer using the pointer IBUGI which is then updated to point to the next free space available in the buffer.

If the character input buffer has more than 9 characters stored in it, or if a carriage return character is detected on input, the input wait status bit is cleared, allowing a user's program to be continued at the next exchange point.

Furthermore, if a user's program is only waiting for a single character input as in the case of ASK and MODIFY commands, it is necessary to enable the user to obtain the character immediately. Thus entry into either ASK or MODIFY or new LIBRARY commands cause a single character input mode flag to be set which signifies that the input wait bit must be cleared upon the receipt of a single character.

Exit from the keyboard routine returns control back to the interrupt skip chain after replacing the updated user's parameter back in their "base area" using the Routine OFFDECK.

6.6.4. Printer Service Routine

This routine is also entered from the interrupt processor skip chain, the routine ONDECK being used to obtain the interrupt user status parameter.

The output buffer is examined in order to determine if any characters are available for printing by using the pointer OBUFO. If no characters are present in the output buffer, the printer flag is cleared, and the output wait status bit for this user is also cleared. If a character is found in the buffer, it is transmitted to the printer for printing and the software teletype in progress flag set.

Exit from this routine returns control back to the interrupt skip chain after replacing the user's status parameters with the routine OFFDECK.

6.6.5. Character Printing Routine ACTION Q

This routine is used for the handling of character for output. If the teletype "in-progress" software flag is set, a character is loaded into the output buffer using the pointer OBUFI, the pointer being updated to point to the position in the buffer where the next free space should be. If the teletype in progress flag is not set, the character is sent directly to the teletype and the in progress flag set.

6.6.6. User Swapping Routines

As previously discussed, in a time shared system, effective measures must be taken in order to avoid the possibility of an I/O bound user holding up the execution of another user's program. Also users must be exchanged at other set times so as to give each a fair share of available computing time. In the executive routines of the two user system these operations are carried out by four routines.

6.6.6.1. Character Input Routine to the Interpreter XRD

Any request from a FOCAL program for input from the keyboard device causes program control to be transferred to this routine.

The users status parameter are set up in page zero with the routine ONDECK and the teletype input buffer is examined for a character by using the pointer IBUFO. If a character is found in the buffer, that location of the input buffer is cleared, the pointer IBUFO updated, the status parameters are returned to their "base area" of core by using OFFDECK and the character returned to FOCAL for program continuation.

Alternatively if the input buffer has no available character the users input wait status bit is set, the return address to FOCAL is saved in the location PCM of the users status parameter which are then returned to base area. Control is then passed on to the new user selection routine EXCHE.

6.6.6.2. Character Output Routine from the Interpreter EXPRN

An output request from a FOCAL program causes control to be transferred to this routine. The user's status parameters are initially loaded into page zero by using ONDECK, and the character is loaded into the output buffer or printed by use of routine ACTION Q. If the character is successfully loaded into the buffer or printed, status parameters are returned to base area and program control returned to the active FOCAL

program.

Alternatively, if the output buffer becomes full at this stage, the output wait status bit for this user is set, the return address to FOCAL saved in PCM and status parameter returned to base area. Eventually control is passed on to the new user selection routine EXCHE.

Fig.33 illustrated the method by which all the character handling routines are related to the I/O buffers.

6.6.6.3. User Swapping Points Check Routine EXCHK

This routine is entered at the start of every sub line break point of FOCAL. A counter in this routine maintains a record of the number of lines executed in the active user's program. If the requisite number of lines have been completed (this number may be changed in order to tune the system) the return address in FOCAL for this user is saved and control is transferred to EXCHE.

6.6.6.4. New User Selection Routine EXCHE

This routine is used for testing the availability of either of the two system users. Any user in input or output wait status is not allowed to become active. The wait status bits of the users are therefore tested until a user is found in the requisite state for becoming active.

If the selected user was the last active user, control is returned immediately to the continuation point in FOCAL. Alternatively, if the selected user was in a non-active state, the program parameters of the two users are exchanged so as to set the selected user in a active status. On completion of the data exchange process, control is returned to the correct continuation point in FOCAL.

6.7. Summary

All the routines were written using the available PAL III cross assembler system so as to produce a standard binary overlay tape for 4K FOCAL 1969. The listings have been included on microfiche and supporting documentation in terms of flowsheets in appendix E.

The remainder of this section deals with the modifications that were necessary to allow the interpreter to operate under the new core configuration and the differences between the real-time extensions

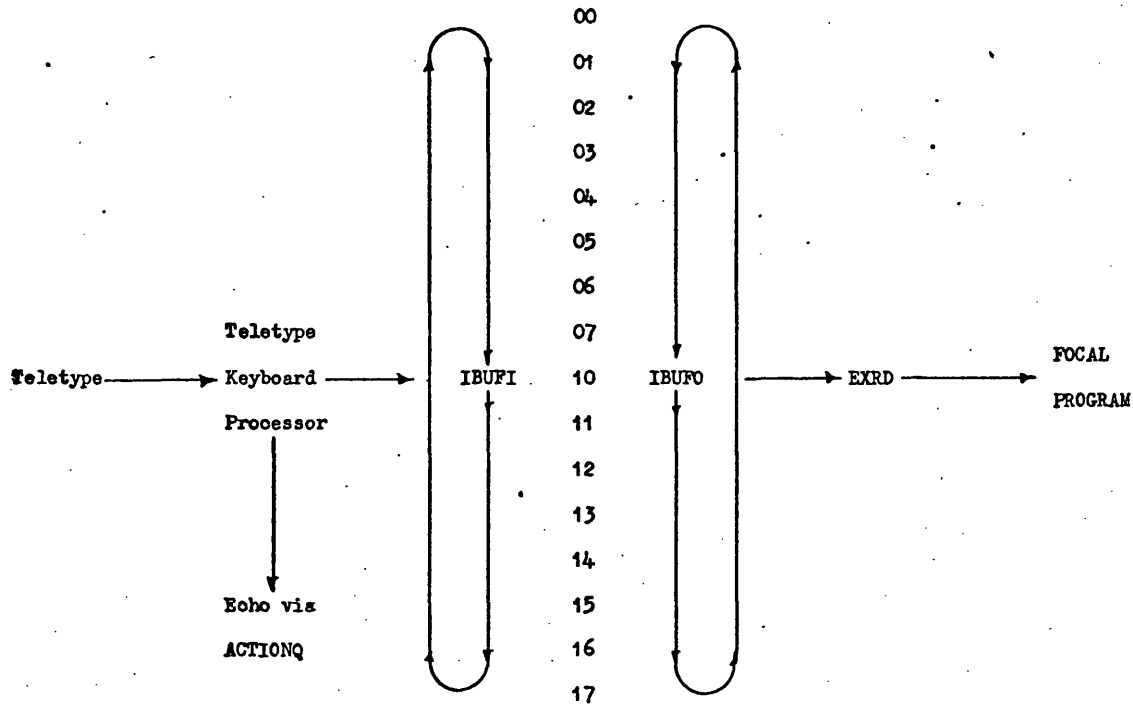
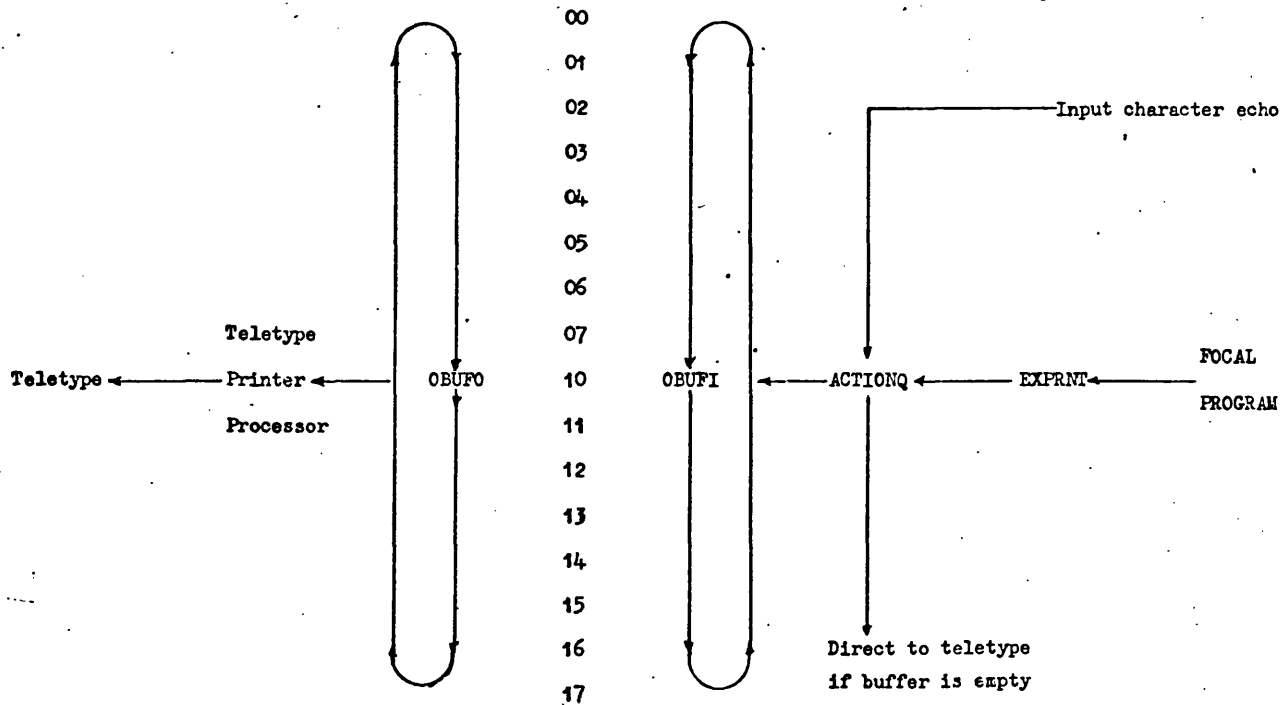
Teletype Input BufferTeletype Output Buffer

Figure 33. Operation of Teletype Input and Output Character Buffers

0	1	2	3	4	5	6	7	8	9	10	11
	INPUT WAIT	OUTPUT WAIT	CTRL/S TYPED	CTRL/S READY			ECHO DISABLE			TTY IN PROGRESS	SINGLE MODE

BIT 1: Input wait bit

The bit is set to a 1 or wait status if the users program is waiting for a character to be input.

The bit is cleared for non wait status if more than 9 locations of the input buffer have been used or if a RETURN is detected on input. If the user is in single character mode (see BIT 11) this wait bit is cleared after the input of a single character

BIT 2: Output wait bit

The bit is set to a 1 or wait status if the users program sends a character to the output routine and encounters an almost full buffer.

The bit is cleared for non wait status if the character output routine finds an empty buffer

BITS 3&4: Used in the On-line Parameter Modification Routine

If a CTRL/S character is detected on input, BIT 3 is set to a 1. When the next RETURN is detected BIT 3 is cleared and BIT 4 is set to a 1 to denote that the message has been entered. BIT 4 is cleared once the message has been decoded

BIT 7: Echo disable bit

Set by CTRL/R code in order to kill input echo. Cleared by CTRL/C or CTRL/T

BIT 10: Teletype in progress flag

Set by the character output routine when a character is sent to a teletype. Used by the output routine to determine whether a character should be sent directly to the teletype or to the output buffer. Cleared by having an empty buffer and the teletype done flag being set

BIT 11: Single character mode flag

Set by ASK or MODIFY commands. Causes the input wait bit to be cleared after every input character. It is cleared whenever a new user is brought into active status

necessary for single user FOCAL and those necessary for two user FOCAL.

6.8. Modifications to the interpreter to allow for proposed core configuration

The proposed core configuration is as illustrated in Fig.31 and it is similar in many ways to that of 8K single user system. Except for the obvious fact that the storage area is shared between two users and executive routines have been included so as to time share the interpreter, the only difference is that the interpreter is now resident in field 1 instead of field 0.

Therefore, modifications similar to those made in the single user version (Chapter 5) to account for the change in variable search routines, text storage and access routines, and push down list controls would have had to be made to the two user versions, the only difference being the change in field setting and the location within the interpreter.

6.8.1. LIBRARY Command

To the user this command is identical in operation to the LIBRARY command in the 8K version of FOCAL, each user being able to allocate his fixed storage area between text and variables.

It was however necessary to add extra protection facilities so as to avoid the possibility of one user straying into another user's area.

6.8.2. Symbol table Editing and Saving

The commands used for editing symbol tables and saving them are identical to those commands used in the single user version in both operation and implementation.

6.8.3. MODIFY command

The modification to single user real time FOCAL whereby a user may duplicate lines has also been included in the two user version. The modes of operation and implementation are identical in the two versions.

6.8.4. Hard Copy for ASK

The hard copy facility implemented on the ASK command in single user FOCAL (i.e. line feed response to the ASK prompt) has been implemented on the two user version in an identical fashion.

6.8.5. Random Number Generator FRAN

A random number generator of the same form as that used in the single user versions was implemented in the two user version in an identical fashion.

6.8.6. Other Differences

The high speed reader was made unavailable because most of the system terminals were likely to be located in laboratories remote from the computer and users would not therefore have easy access to this facility.

6.9. Timing Considerations

Before proceeding to develop timing and synchronous data sampling facilities in an identical fashion to those produced in the single user real-time FOCAL, it was again necessary to consider the possible timing errors which could occur.

For example, assuming that similar timing functions existed, the following could represent timing loops created by both of the system users.

<u>USER 1.</u>		<u>USER 2.</u>	
11.01	I (FLAG (A)) 11.10	11.01	I (FLAG (A)) 11.10
11.02	S I = I + 1	11.02	S I = I + 1
11.03	S J = J + 1	11.03	S J = J + 1
11.04	G 11.01	11.04	G 11.01
11.10	T " TICK " , !	11.10	T " TOCK " , !
11.12	S J = 0 ; S I = 0 ; G 11.01	11.12	S I = 0 ; S J = 0 ; G 11.01

Under unfavourable circumstances, any user's scan flag could become set immediately after it has been interrogated. If users are being swapped every four lines and this user has only just been set up in an active status, then three lines of his own program and four lines of the other users program might have to be executed before the flag could be re-interrogated.

Assuming an average time of 7 m seconds per command, together with the 12 m seconds required for user swapping, a delay of up to 70 m seconds could occur. This of course is the worst possible case as it is unlikely that four commands would be included within a simple timing wait loop. It could also be reduced by swapping users more frequently, although this would increase the proportion of processing time used for swapping.

However, if one considers a random error in timing of between 0 and 70 m seconds, results of the simulation study outlined in Appendix B would appear to show that this magnitude of error is acceptable provided that sampling periods of not less than $\frac{1}{4}$ second are used.

Thus it was decided to implement scanning facilities in the two user version in an identical manner to the scanning procedure implemented in the single user version. Synchronous data sampling being set up by using a software flag facility and I/O peripherals being directly accessed rather than clock driven (except for the Input and Output Counter Card Systems).

As the computer system is only provided with a single clock whose interrupts have to control the timing properties of the operating system, it was necessary to implement the timing facilities in the following manner so as to prevent one user corrupting the timing properties of another user program.

As the users will not necessarily require identical scanning periods, it was necessary to provide each user with his own resettable scan flags. As both sets of scan flag counters would be controlled by clock interrupts, the clock rate would have to be preset with users unable to alter the frequency of timing pulses.

In the single user version of Real-Time FOCAL, the facility was provided whereby the user could set up the value of his absolute time counters. This could have been included in the two user version, if each user was provided with his own set of time counters. Having to update two sets of current time counters each time there was a clock interrupt, results in a waste of processing time and core space which cannot be afforded in a time shared system.

It was therefore arranged that only a single set of absolute time counters would be made available. The system users would be allowed to share the facility but not allowed to reset the clock counters. If absolute time is required then it would be necessary to preset the clock counter on

initialization.

Thus the FTIM and FLAG functions have been implemented and are available in the form shown in Fig.35.

6.10. Priority Tasking System

A foreground/background priority tasking system has been adopted in the two user version of Real-Time FOCAL. The system uses the end of line break point so as to avoid confusion with the beginning of line break point which was allocated to user swapping.

Group 31 has been allocated as the priority group. Each user has been provided with identical facilities for enabling and disabling the priority group as were provided in the single user versions, i.e. the E C and E D commands.

6.11. Input/Output Transfer

All the input functions produced are identical in operation and implementation to those used in the single user version. This was possible because there was no need to protect against one user reading another user's input device. Each time the peripheral is accessed, the current value is obtained. Even in the case of the input counter card system, the data can be re-accessed as it is not cleared after reading from the input data table.

Output functions on the other hand must be protected so that one user cannot transmit data on another user's peripheral devices.

Protection has been achieved by allowing one user to output on even channels only and the other user on odd channels only. The channel address arguments of output functions must therefore be examined and compared with the allocated channel addresses for the active user.

As this form of protection must be applied to all output functions, it was decided that the "manual" incremental output function FINC should be incorporated within the FOUT command as

FOUT (0INC , CHANNEL , VALUE)

so that the protection routines added to the FOUT command could be used.

The remainder of the output functions are identical in nature to those used in the single user versions of Real-Time FOCAL.

6.12. Control

The PCI control algorithm was adopted for the two user system but implemented in a slightly different form. Instead of using a FOCAL function call, the control algorithm was written as a command. The reason for this was that a slightly improved speed of operation would result. Also the available space within the same field as the interpreter meant that all extended functions could be retained and the addition of more function calls would have required a re-arrangement of the function sort and branch tables. The alternative procedure of using an available space within the command sort and branch lists (originally used for the high speed reader) was a more attractive proposition.

The command was developed to be used in the following forms:-

PCI (A , B , C ; D , E , F ; G)

PCI (A , B , C , D , E , F , G)

where A is the output channel address between 0 and 9

B is the loop measured value

C is the loop set point

D is the Integral Action time

E is the Scanning interval

F is the Gain Factor

G is a named variable which is given the value of the
computer incremental output.

The final argument has been included so that a record is available of the incremental output value for the purposes of either checking the numerical value produced by the control algorithm or for the purposes of cascade control. In cascade control, the output of one control block would be used as a variable parameter for another block. As no actual output to the equipment would be required under these circumstances the ; delimiter has been included before the final argument so that output to the output counter card system may be inhibited.

Another useful feature of the ; version of the command is that it may be used to initiate the control algorithm. Inhibiting output would avoid the initial "bump" in the system when the control algorithm is started.

The use of a single channel address / loop number parameter is restrictive but was a necessary condition which had to be imposed upon the system so as to provide user protection and also to reduce the time taken by the control algorithm. One user has been allowed even channel addresses and the other odd channel addresses. It was found that this condition could be implemented easily if loop parameters were stored as a combined table rather than having a separate table for each user.

The system adopted also meant that one less argument was required thereby saving a little time in evaluation on manipulation routines.

The algorithm is completed by calls to a floating point interpreter, using the last argument as a variable name into which the floating point value may be loaded using the variable search and save routines available within the FOCAL interpreter.

The computed value is also converted to integer which is clocked out every second by the interrupt processor in an identical manner to that used in the single user version of FOCAL.

6.13. Parameter Modification Routine

To the external user, this routine is identical in operation to the routine used in the single user version of FOCAL. However as the modes of character handling in the two systems are different, it was necessary to adopt a somewhat different approach.

When a CTRL/S control code is detected by the keyboard processor, it has been arranged that a flag is set within the user's status parameter (see Fig.34) to signify that the next set of input characters is to be used for parameter modification. At this point it was also found necessary to inhibit TYPE and ASK statements but only for the user trying to modify one of his parameters. A general inhibit in TYPE and ASK would cause the other user to be held up as well.

When all the flags and inhibit switches have been set, a > character is echoed by the computer as an acknowledgment.

It has been arranged that subsequent characters typed in at the keyboard, are loaded into the character input buffer and echoed on the printer device. When a carriage return character has been detected, another flag is set in the user's status parameter signifying that the complete command has been entered into the buffer.

Using the end of line break point, the command complete flag is examined at the end of every line in the FOCAL program. When found in the set condition, FOCAL's internal routines are used to get the input characters and perform a forced SET command. Completion of this SET command results in the flags and inhibit switches being cleared and a < character is echoed on the terminal printer to signify completion of the task.

If an ASK or TYPE statement is executed during the process of parameter change, control is passed immediately to the handling routine to wait for the necessary input, the user being put into an input wait status until all the characters of the command have been entered into the keyboard input buffer and thereby causing no delay to the other system user. After the direct SET command has been forced, the inhibited ASK or TYPE command is executed.

6.14. Error Code

It was necessary to provide facilities for trapping out erroneous conditions during the execution of all the new functions. This was accomplished by standard calls to the error recovery routine. A list of error codes produced has been tabulated in Fig.37.

6.15. Initial Dialogue

In order to avoid users corrupting each other's timing properties, they could not be permitted to alter the clock interrupt rate or absolute time counters.

These could either be preset within the Assembler overlay program or else facilities provided whereby the values could be selected at loading time.

As different users might require different amounts of core, it would be very useful if at system loading time, core allocations could be selected for each of the two users.

Thus an "Initial Dialogue" section was included so that these parameters could be set up easily from the teletype terminal. Fig.36 illustrates the use of the initial dialogue.

6.16. Conclusions

The resulting core configuration of the 8K system is as shown in Fig.38, and it can be seen that it was possible to include all the standard extended FOCAL functions within the system.

This two user version is still essentially undergoing a period of trials, experience to date suggesting that the system is capable of providing two users with extensive facilities for data acquisition and control. The main disadvantage of the system, which was realised at the outset, is one of maximum data rates which the system is capable of. However, most of the laboratory applications so far could be easily dealt with, even when a maximum clock rate of 4 interrupts per second has been imposed.

Again, its main advantage arises from the interpretive mode of operation, allowing programs to be readily created and modified. If this is an essential requirement of a computer operating system, then the slow speed of response must be expected.

FUNCTIONS

Single User

S Z=FTIM(OST,A,B,C,D)

S Z=FTIM(X)

S Z=FLAG(OST,A,B,C,D)

S Z=FLAG(B)

S Z=FIN(OHRZ,A)

S Z=FIN(OADC,A)

S Z=FIN(ODPM,A)

S Z=FIN(ODIG,A,B)

S Z=FOUT(ODAC,A,B)

S Z=FOUT(ODIG,A,B,C)

S Z=FOUT(OALM,A,B,C)

S Z=FINC(A,B)

Two User

Not Available

All available
in identical
form

S Z=FLAG(OST,B,C,D)

Available in
identical form

All available
in identical
form

All available
in identical
form

S Z=FOUT(OINC,A,B)

REASON FOR DIFFERENCES

Only a single set of clock registers is used in the two user version. Users cannot therefore be allowed to reset the registers which are set up in the Initial Dialogue.

The argument for the number of clock pulses is not required as it is set in the Initial Dialogue so as to avoid corruption.

The only difference in the output functions is that user 1 has been allocated even channels and user 2 odd channels so as to avoid data corruption.

This has been incorporated within the FOUT function so as to make use of the protection facilities provided for within the FOUT function.

Figure 35. Differences Between Single User and Two User FOCAL

<u>FUNCTIONS</u>		<u>REASONS FOR DIFFERENCES</u>	
<u>Single User</u>	<u>Two User</u>		
S Z=FCON(A,B,C,D,E,F,G)	PCI(A,B,C,D,E,F,G)	Implemented as a command rather than as a function so as to improve the execution time. Output protection facilities have also been included in the command	
S Z=FDYN(A,B,C,D,E)	Not available	This function has been left out because of limited space and limited use of this particular function.	

Figure 35. Continued

```

HELLO FOLKS:
END OF USER 1 PDL OCTAL 3000-6600:7700
HELLO FOLKS:
END OF USER 1 PDL OCTAL 3000-6600:1234
HELLO FOLKS:
END OF USER 1 PDL OCTAL 3000-6600:5400

WHAT IS THE TIME
SECS:65
WHAT IS THE TIME
SECS:00
MINS:40
HOURS:12
DAYS:19
CLOCK PULSES PER SECOND,1,2 OR 4:27
CLOCK PULSES PER SECOND,1,2 OR 4:01
THANKS
:700.00
*
```

Figure 36. Example of the use of the Initial Dialogue for Duo

<u>Error Code</u>	<u>Diagnostic</u>
11.73	Illegal sub-function call in FIN or FOUT function
11.;9	Illegal channel address in FOUT function call. User 1 even, user 2 odd
13.57	Exit from LIBRARY command
13.90	Terminator other than , or ; or RETURN in variable erase command E F,
15.13	Illegal character in multiple argument handler
15.23	Missing argument in function call
15.48	No scan flag of that value has been set up
15.56	Too many arguments in FTIM or FLAG function
15.77	Illegal code in FTIM function
16.03	PCI command should begin with a bracket
16.07	Channel address should be positive
16.18	Illegal channel address in PCI (0 to 9 only are legal)
17.06	Only ; or RETURN are legal terminators of PCI command
17.<7	Too many arguments in FIN or FOUT function call
21.<4	Illegal character or no such line in MODIFY command
25.92	Character other than A, C or D in ERASE command

Figure 37. Error Diagnostics for Two User FOCAL

DUO Core map Field 0

0	Zero Page Pointers		Automatic Restart Routine	
1	Interrupt Processor			
2	Keyboard Processor			
3	Onlock, Offlock and Dump		List	
4	List	Action	Printer Processor	Lock1 Lock2 Free
5	Exornt	Exrd	Exchk	User Swapping Routines
6	Free		User 1 I/O Buffers	User 2 I/O Buffers
7	Ctrl/s Extension	Synchronous Service Routines		Free
10	Non-active Users Saved Pointers			
11	User 1 Command Input Buffer		User 2 Command Input Buffer	
12				
13				
14				
15	USER 1			
16	TEXT VARIABLES AND PUSHDOWN LIST			
17				
20				
21				
22				
23				
24				
25				
26				
27	USER 2			
30	TEXT VARIABLES AND PUSHDOWN LIST			
31				
32				
33				
34				
35				
36				
37				

DUO Core map field 1

0	Zero Page Pointers				
1	Command Decoder			Getln	
2	Entabf		Do	Pushdown List Controls	
3	Control & Transfer		Write	Testc	Sortc
4	Conlst	If	Set & For	Dys & Int	Conro
5	Type & Ask		Modify	Sortj	Adc, Outl and others
6	Getarg			Spac, Testn, Ren, Parj and others	
7	Ecall				
10	Termc	Egn, Abs et al	Testlpr, Partest	Delete	Readc
11	Erase		Findln	Getc	Endln
12	I33, Prmtln, Prnt and Printc			Packc	Extensions
13	I/O Routines, swapping Routines, etc. to High Level, Single Form, etc. user recovery, etc.				
14	Rubout		Symbol Table Dump	Extensions, spm, etc. etc.	
15	Extensions to Library and Variable Search Routines			Variable access	
16	Parameter Modification Routine and Search routine				Lists
17	Argument Evaluation		Flag	Ptin	Std. Print
20	Control Program				
21	Pin and Pout Commands				
22	Tran		PCI Store Table		
23	Exp		Atn		
24	Log				
25	Cos and Sin				Volume mode
26	FLOATING POINT				
27	INPUT AND OUTPUT				
30	ROUTINES				
31	I/O and S.D.				
32					
33	FLOATING POINT INTERRUPT				
34					
35					
36	Sgt			Library	
37	Loaders				

Figure 38.

CHAPTER 7.

A File Structured FOCAL Operating System

7.1. Introduction

Having made the decision to develop a file handling system for the reasons described previously, it was necessary to examine the possible ways of providing such a system.

Basically a system was required which would allow for the manipulation of FOCAL program files, FOCAL data files and FOCAL system software, the prime design consideration being that the system should be easy to use. Such a system could have been achieved in any of a number of ways:-

1. By modifying and enhancing existing manufacturers software, OS8, so that FOCAL software could be implemented on the system.
2. By further developments to FOCAL, enabling FOCAL software to be saved as files on magnetic tape from commands or functions within FOCAL.
3. By developing an interactive file MONITOR system, accessible from FOCAL, allowing all forms of FOCAL software to be saved on magnetic tape.

The eventual choice of system would be dependent upon factors such as the degree of flexibility provided, the amount of programming effort required to implement the system and whether the system in its final form would be easy to use.

It was decided that extending OS8 so that FOCAL could be implemented within the system would not be a viable proposition. Experience suggested that the OS8 system was by no means easy to use because of the extensive facilities which it provides. Implementation of FOCAL within the system would require restructuring of the existing FOCAL interpreter in order to make FOCAL compatible with the device handling and file manipulation techniques utilized in the OS8 system. This would also have necessitated an in depth analysis of the OS8 operating system in order to determine the nature of the routines used. Although a very flexible system could eventually be produced, it was felt that an excessive amount of effort would be required and the final system would not necessarily be easy to use.

The second method could have provided a suitable basis for a FOCAL file handling system, The extra 4K of core available in addition to the 8K

required for the FOCAL system would provide sufficient core space for the new functions and commands required for the saving and loading of program and data files. Such a system would be essentially very easy to use as only a few extra commands would be implemented within the FOCAL system. However the major drawbacks of such a system would be the limited flexibility provided by the system and the difficulty which would be encountered in the saving and loading of different versions of the FOCAL interpreter. These factors could be overcome with suitable programming effort which would however result in the system becoming more complicated to use.

The third alternative of providing a stand alone monitor system was therefore considered to be the method which would provide an ideal solution. By basing the monitor system upon simple and easy to use commands of a form similar to those used in FOCAL, a system could be devised in which any form of FOCAL software could be saved as files. This solution would only require minimal modifications to any FOCAL system so as to enable linkage between FOCAL and the monitor system from a single FOCAL command. As FOCAL system files would essentially be a direct core image stored on magnetic tape, the monitor system could easily be extended to provide facilities for saving image files of executable binary programs. This would provide an extremely useful facility for experienced programmers.

It was therefore decided that the monitor type of system should be implemented.

7.2. Structure of a File Monitoring System

The structure of the proposed monitor system was essentially dependent upon whether it was thought necessary to extend the available 8K FOCAL system to make use of the additional 4K of memory in the 12K PDP8-E.

If FOCAL was to be kept as an 8K operating system, the monitor could be developed as a core resident system, utilizing the 4K of memory not occupied by the FOCAL system.

Alternatively, if some of the extra 4K was used to provide additional space for program and variable storage, it would have been

difficult to provide sufficient space for a totally core resident monitor. In this case, only the magnetic tape handling routines, terminal handling routines, and a command decoder would remain resident in core, a similar type of structure to OS8, software associated with the commands of the system being stored on tape and read into core when required for operation.

Experience which had been gained when 8K FOCAL Extensions (Chapter 4) (64) were made available during a final year undergraduate design project showed that a limited program size with the ability to chain modular programs together could be highly advantageous. Developing programs in this form meant that it was essential to check computed data after each program module. Non sensible data could therefore be detected early on in the programming chain and errors in method, program or input data corrected before proceeding on to the next stage.

The alternative procedure, adopted by some students, of developing one large program to perform the same task using FORTRAN on the University's computer, invariably took far longer to get the program into an operational form.

On this basis, it was decided that the 8K FOCAL operating system provided sufficient program storage as it stood and it was therefore unnecessary to extend program storage into the extra 4K of memory.

The monitor system could therefore be designed as a core resident system in the upper 4K field of the 12K system.

7.3. Requirements of the Monitor System

The most essential feature of any file handling system, is the ability to select by name, files stored on a particular magnetic storage device. This requires a directory of all files to be maintained for each magnetic storage device. Data regarding the position of the file on the device, the length of the file and its operational location in core, must be stored in the directory under the name of the file.

When a file is called for loading into core, the information in the directory can then be used to locate the file on the device and load it into the required position in core.

Similarly when saving a file, the file must be written on to the device and a record of its name, location on the device, and core location must be inserted in the directory.

The directory therefore not only maintains a record of file data but also a record of the amount of space still available on the device.

Other features which would form the essential requirements of any file handling system are:-

1. The ability to list the directory of any device so that a user can determine if a particular file is available on that device
2. The ability to save named files of three types on any storage device:-
 - a. FOCAL program files
 - b. FOCAL data files
 - c. Core image binary files of either FOCAL system software or other executable binary files.
3. The ability to load into core named files of the above three classes from any device.
4. The ability to delete any named file from any named device and to recover the vacant space efficiently.
5. The ability to run any file, which has been loaded into core, from a particular starting address.

Other features which could be incorporated into a file handling system so as to extend its capabilities and ease of use would be:-

6. The ability to copy a named file from one device to another or on to the same device. With only the basic minimum features, this would have to be done by loading the named file into core and then saving it again.
7. The ability to search the directory of a specified device to determine if a particular named file was present on that device. This would avoid the need for listing the whole of the directory to find if a specific file was present on the device.
8. The ability to completely erase all files from a specified device rather than having to delete each file sequentially. This particular process would have to be used carefully and have sufficient error detection procedures so as to avoid the possibility of deleting important files through misuse of the command.

Most of these particular facilities were dependent upon the ability to program data transfers between core memory and the dual drive DECTape magnetic storage devices available.

It was therefore necessary to examine the operation of the DECTape system and devise a general purpose controlling routine capable of performing the appropriate data transfers required.

7.4. DECTape Programming

Unlike most other common forms of magnetic tape storage devices, the DECTape system stores information at fixed positions on magnetic tape rather than at variable positions. This feature allows the replacement of blocks of data on tape in random fashion without disturbing data previously recorded in other blocks.

This block type structure of the DECTape is made possible by using formatted magnetic tapes upon which mark and timing information has been pre-recorded. When reading from or writing to magnetic tape the format and timing information is used to determine the exact position on the tape of the transfer.

Each formatted tape is segmented into a set of sequentially numbered blocks, the length of each block being selected in the formatting stage. All blocks on the tape can be made either the same length or of different lengths depending upon the requirements of the system.

The only constraints upon the size of a block is that the number of words in a block must be an integer multiple of three. This is because the data transfer operations are controlled by the mark track information which requires the reading of six full lines from tape before a complete code can be read from the command register. Normal data is stored on the magnetic tape at 3 bits per line and 4 lines per word. Thus the lowest common multiple of the six line mark track record and the four line data record is twelve lines, the equivalent of three twelve bit words.

The operation of both DECTape drives is controlled by the use of four flip-flops which can be set under program control to select:-

- i. The required drive
- ii. The direction of motion of the selected unit
- iii. Whether to start or stop the selected unit

- iv. Whether data is to be read from or written to the selected unit

The flip-flops are set as shown in Fig.39, by loading the device command register.

Mark track information can be obtained from the tape, by reading the command register back into the accumulator, Fig.40. Each time a new line is read from the DECTape device selected, a new bit is rotated into the least significant bit of the mark track register, the most significant bit of the register being lost. A single line flag is set by the controller at this point to signify that another line has been read. Meaningful mark track codes are as shown in Fig.41.

As data is stored as four lines of three bits, a QUAD line flag is set for each four lines of mark track data which is read from tape.

7.5.1. DECTape Controlling Routine for the Monitor System

This routine is the most important routine of the file monitor system as it must control transfers of data between core and magnetic tape effectively and with a minimal possibility of data corruption.

Manufacturers software for controlling data transfers to and from the DECTape with checksum error protection, was available as a subroutine which could be incorporated into user written software. This however had a number of disadvantages which had to be overcome before it could be adopted as a handler in the monitor system.

The existing software allowed for the transfer of up to 32 blocks of sequential data to or from either DECTape drive, each block of data being 128 words in length (with the option of 129 words), data transfers of less than 128 words not being allowed, and automatic continuation of transfer into the next field of memory not being provided for.

The system of fixed block transfers was an obvious necessity if data transfers to and from tape were to proceed without affecting other data stored on tape.

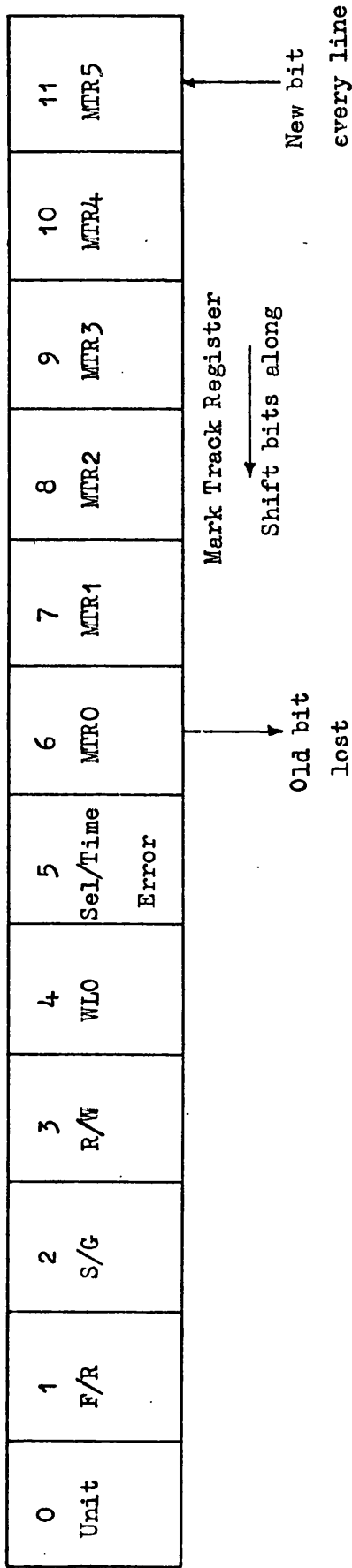
The internal configuration of the PDP-8E segments a core field into "pages" of 128 words in length for addressing purposes. A 129 word block format on tape would therefore be a logical choice, this being the nearest number of words per block in which a magnetic tape can be formatted

0	1	2	3	4	5	6	7	8	9	10	11
Unit	F/R	S/G	R/W								

not used to load the Command Register

- Unit Selects which drive is to be used for a data transfer
 0 = Unit 0 of Dual Drive System selected 1 = Unit 1 of Dual Drive System selected
- F/R Determines the direction of motion of the selected drive
 0 = Forward
 1 = Reverse
- S/G Instructs the selected drive to move or stop the tape
 0 = Stop
 1 = Move
- R/W Instructs the selected drive to read data from or write data to the tape
 0 = Read from tape
 1 = Write to tape

Figure 39. Format of the Word Loaded into the Command Register from the Accumulator



WLO Unit selected for data transfer to was not write enabled if bit is set to a 1

Sel/Time

If bit is set to a 1 it indicates one of the following errors

Error

a. No unit has been selected for a data transfer

b. More than one unit has been selected for a data transfer

c. The control program was not operating fast enough to return to work on the Data Register before the next transfer of the DECTape took place

MTR0

to Six bit Mark Track Register

MTR5

Figure 40. Format of the Word Transferred from the Command Register to the Accumulator

Figure 41. Block Structure of a Formatted DECTape

so as to correspond to the 128 word page structure adopted on the PDP-8E. However having to always transfer a whole block of information would have been a potential source of corruption when saving and reloading FOCAL programs. For instance, if a data set had been created by one program and the monitor system was then used to load in another program for further data processing, the start of the symbol table could easily be overwritten.

There was a need therefore to provide a system whereby a part of a block could be transferred to or from the magnetic tape. It was therefore arranged that if only a part of a block was required, the remainder of the block would be filled with zero data for convenience in computing checksums. On re-reading such a block back into core, only the meaningful data would be used, the zero data being read only and used for the computation of checksums for comparison purposes. This required an extra parameter for the handling routine to signify the fact that only part of the block was required.

The feature whereby the system only allows a maximum data transfer of 32 blocks, with no cross field transfers, was thought to be restricting, particularly in the case of storing FOCAL system software where a 64 block transfer would be required, with the ability to change from one field to the next.

It was therefore arranged that if the address pointer of the subroutine returned to 0000 at any stage, the field setting would be incremented to the next highest field. Also, transfers of greater than 32 blocks were arranged by modifying the method by which the subroutine was supplied with transfer control data.

Another facility which was thought to be particularly useful was the ability to test high priority data transfers for integrity by checking data stored on tape with data stored in core immediately after a transfer had been completed.

This was accomplished by providing a branch within the section of the routine associated with reading data from magnetic tape. Instead of storing the data in core, the data read from tape is checked against the data already in the particular core location in question. As this system was only to be used for high priority data, enablement and disablement of the branch must be accomplished by patching in an instruction by software

immediately before and after the controlling routine is used.

A facility was also incorporated whereby a software switch could be set, thereby allowing the routine to search for a particular block on tape without reading it. This provided the ability to reposition a tape after a data transfer had been completed.

7.5.2. Use of the Controlling Routine

For a subroutine developed with the above facilities, the transfer of a file to or from a magnetic tape required that the following parameters be defined to control the transfer operation.

1. The DECTape unit of the dual drive system which is to be involved in the transfer
2. The direction of transfer, i.e. is it read from DECTape into core or write from core on to DECTape.
3. The block number on the DECTape in which the file starts.
4. The initial field of core in which the file is to be transferred to or from, this value being changed automatically for cross field transfer.
5. The number of complete blocks on tape occupied by the file.
6. The number of meaningful data words in the final block of the file.
7. The starting address of the file in the specified field.
8. The direction in which the selected drive should start, i.e. forward direction or reverse direction.
9. A switch to select either search mode or transfer mode.

For convenience, it was arranged that these parameters should reside in Page zero of field 2 as shown in Fig.42 so that they could be accessed directly from any position in core.

As the subroutine would also have to be accessed from any position within the field in which the monitor system was resident, the most convenient method of linking to the routine would be via a pointer located within zero page.

If an error occurs at some point during a transfer operation, for example, attempting to write to a write locked device, it is necessary to signal the error condition in some fashion. This has been achieved by providing a multiple subroutine return. An error condition returning to

<u>Location</u>	<u>Contents</u>	<u>Function</u>
0057	UNIT	Set to 0000 for unit 0 and 4000 for unit 1
0060	REDWRT	Set to 0000 for Read and 0001 for Write
0061	BLKADR	Starting block for DECTape transfer
0062	FIELD8	Field of transfer (0000 to 0007)
0063	BLOCKS	Number of whole blocks to be transferred
0064	PRTBKS	Number of words used in final block
0065	BUFADR	Core address for transfer
0066	FWDRVS	Set to 0000 for forward and 0001 for reverse
0067	MWORDS	Reset counter for number of words per block
0070	WCOUNT	Counter for number of words per block
0071	PWORDS	Counter for number of words in final block
0072	BFCNT	Pointer for transfer from tape to core
0073	P10	Constant used for field change process
0074	BLANK	Constant used for field change process
0075	SRCHWD	Set to 0001 if search only is required

Figure 42. Parameters used in DECTape Controlling Routine

```

CLA CLL           / Set parameters to read from DECTape on unit 0
DCA UNIT          / into core.
DCA REDWRT        / Transfer one block of 200 words into field 0
DCA FIELD8        / starting at address 0000 from block 100
IAC              / of the specified DECTape
DCA BLOCKS        /
DCA PRTBKS        /
DCA BUFADR        /
DCA FWDRVS        / Set initial motion of tape in the forward direction
DCA SRCHWD        /
TAD C100          /
DCA BLKADR        /
CLA CLL           /
DTAPE            /Call controlling subroutine
ERROR            /Return to this location signifies transfer error
CLA CLL          /Continue

```

C100, 0100

Figure 43. Use of DECTape Controlling Routine

the first statement after the calling statement and a correct transfer returning to the second statement after the calling statement.

Thus the sequence for using the control routine is as follows:-

1. Set up the zero page parameters for file transfer
2. Call the subroutine
3. If the first return is taken, signal an error condition and if the second return is taken, continue with the next sequence.

Fig.43 shows an example of the use of the routine within an Assembly Code program.

The subroutine was written in PAL III assembler code and occupied location 1000 through 1377 of field 2 of the monitor system program.

7.6. File Directory System

As mentioned previously, it is essential to keep a directory of files present upon a particular device. This meant that it was necessary to define an area on each tape which could be used to store the list of files. The size of the area, would be dependent upon the file information required, the method of naming files, and also the maximum number of files which were likely to be accumulated upon a particular magnetic tape.

The size of the directory area would also affect the methods in which a directory could be searched in order to find a particular named file. As it would be necessary to read the directory into a buffer for a file search procedure, a long directory would necessitate that the directory area would have to be read into the buffer in segments. Each segment would then have to be searched in turn for the required file. Alternatively, if only a short directory was necessary, then the whole process could be done in one operation.

7.6.1. Naming of Files

It was decided that files should be named as in the OS8 operating system so as to make the differences between the use of the two systems minimal.

A file name was therefore allowed to be up to six alphanumeric

characters in length, always starting with an alphabetic character, followed by a two alphabetic character file extension, e.g.

ABC . PR

ABC123 . DA

X2Y4Z3 . BN

The extension characters are used to denote the type of file being dealt with. In the monitor system, the types of files to be allowed on the system were,

1. FOCAL program files using the extension character PR
2. FOCAL data files using the extension character DA
3. FOCAL system files or other executable binary core image files using the extension BN.

The file extension is therefore a convenient method of informing the monitor system how a file should be loaded into core.

By storing the file names in the directory as stripped packed ASCII characters, four twelve bit words would be required to hold the six file name characters and two file extension characters.

7.6.2. File Data stored in Directory

It was necessary to define what information about the file should be saved within the directory.

One alternative was to store data regarding the position of the file on the tape and the number of blocks occupied. An initial block in the file area could then contain data regarding the running of that particular file.

Another alternative would be to store all the data regarding position on tape and running of the file within the directory.

The choice of method would be dependent upon the amount of information required for running the file. As it was envisaged that all files would be saved as direct core image files to be reloaded back into the same position in core from which it came, the only information required about loading the file would be the location in core and field from which the file was saved. Admittedly in the case of saved executable binary files the starting address could be an advantage but only of use in a load and go system. As most of the file manipulation was to be with FOCAL

program and data files, this information was not really necessary.

It was therefore decided that all file information should be saved within the directory area of the tape. This required a total of 9 twelve bit words per file, made up of 4 for the file name and extension, 1 for starting block on the tape, 1 for the number of whole blocks occupied by the file, 1 for the number of words occupied in the last block of the file, 1 for the field into which the file is to be loaded and finally, 1 for the address from which the file was saved.

7.6.3. Length of Directory

The tape area required for a directory is dependent upon the maximum number of files which can be saved upon any one magnetic tape.

It was envisaged that most files saved would be of FOCAL programs. The maximum length of a FOCAL program is approximately 7400_8 words in length which is equivalent to 30_{10} blocks of 128 words. The average length of a FOCAL file will in general be much shorter than this. As each magnetic tape can accommodate approximately 1400 blocks of 129 words each, and assuming that the average length of a file will be in the region of 10 blocks, the directory will have to be of sufficient size to accommodate about 140 file names. At nine words per file a directory ten blocks in length would be required.

A buffer for a directory this size could easily be accommodated within the core configuration without the need for segmentation. If necessary, an area in core already occupied could be utilized by the directory by using a swapping procedure in which the active area is first saved on DECtape before reading in the directory, the active area being reinstated after the directory has been used.

7.6.4. Additional Directory Information

In addition to information regarding files it would also be necessary for the directory to keep a record of the next available free block on the tape and the next free location in the directory buffer. This would simplify the process of saving new files on the tape, as a record of where the file was to be added and where in the directory buffer file data was to be stored, would be readily available.

7.6.5. Directory Handling Routine

The directory area is likely to be read from and written to more often than any other section of the tape because of the need to search for files and append new files. It would therefore be preferable to have it located as near as possible to the beginning of the tape, thereby reducing searching time.

Blocks 1 to 10 of each DECTape were therefore selected for the purpose of holding the tape directory.

The position of the buffer used for holding the directory when read into core would be dependent upon the core requirements of the rest of the monitor system. There was, however, sufficient room to locate the buffer in addresses 5000 thru 7400 of field 2, avoiding the need to have a segmented directory system.

A routine was therefore written to perform the task of reading or writing the directory area from tape to core or from core to tape of either DECTape drive using PAL III assembly language.

The routine is called as a subroutine using a pointer located within zeropage of field 2 but first requires that the accumulator and link be set up to define the unit required and the direction of transfer in the following manner :-

```
UNIT 0    set accumulator to 0000
UNIT 1    set accumulator to 4000
Read directory into buffer, set link to 0
Write directory on to tape, set link to 1
```

Fig.44 shows an example of the use of the subroutine from other routines in the system.

7.7. The Interrupt Processor

Operating system communication must take place through the keyboard and printer of a terminal device, which requires an interrupt processor for efficient use. The only other device requiring an interrupt service, is the power failure protection option of the system, the DECTape controller used being a non-interrupt driven device.

Any external interrupt from a device connected to the interrupt skip bus causes the current program operation to be terminated and the

```

CLA CLL          /Set accumulator and link to zero so as to
                  /read the directory into the buffer from
                  /DECTape unit 0
DRCTRY           /Call directory subroutine
.....          /
.....          /
.....          /
.....          /
.....          /Operate on directory and files
.....          /
.....          /
.....          /
.....          /
CLA CLL CML      /Set accumulator to 0000 and link to a 1 so as to
                  /write the directory from the buffer to DECTape
                  /unit 0
DRCTRY           /Call directory subroutine
CLA CLL          /Continue

```

Figure 44. Example of the use of the Directory Handling Routine

equivalent of a subroutine jump to address 0000 of field 0 to be executed, the next sequential address in the terminated program being saved in location 0000.

For efficient use of the system, it would be necessary for all systems software to use one core resident interrupt processor. However, this would involve restructuring FOCAL systems software to make use of such an interrupt processor. Also making the interrupt processor available to all other systems software would increase the possibility of corrupting the file handling system and in particular DECTape transfers.

It was therefore decided that the interrupt processor should be specific to the file handling system.

When linking between perhaps the FOCAL interpreter and the file handling system, it would be necessary to change the address pointer of the interrupt processor, held in location 0001 of field 0, so as to select the correct interrupt processor. Alternatively, the interrupt processor could be made to reside in the first two pages of field 0. This would require that the interrupt processor for the monitor was placed in core whenever the monitor was in operation, and the normal zeropage pointers etc. to be resident when any other system program was operational. Such a technique would ensure that the interrupt processor was never available to other programs but would require the transfer of data to and from tape. Although data transfer are open to corruption, they would be checksum protected. The fact that it would be the interrupt processor which was corrupted first would make it difficult to enter meaningful commands into the system thereby giving an early indication of transfer corruption, avoiding the loss of valuable program files.

It was therefore arranged that the interrupt processor should be placed in the first two pages of field 0 whenever it was core resident, and saved in blocks 11 and 12 of the DECTape on unit 0 when not core resident. Before reading the interrupt processor into core, the program already resident would have to be saved, this being accomplished by writing the first two pages into blocks 13 and 14 of the DECTape on unit 0.

The following sequences would therefore have to be employed when using the system:-

a. Loading a Saved File into Core

1. Save interrupt processor in blocks 11 and 12
2. Load the program file from tape
3. Save program occupying pages 0 and 1 of field 0 in blocks 13 and 14
4. Re-load interrupt processor back into core

b. Executing a Loaded File

1. Save interrupt processor in blocks 11 and 12
2. Re-load blocks 13 and 14 into pages 0 and 1 of field 0
3. Execute program

c. Returning from executing a Loaded File

1. Save program occupying pages 0 and 1 of field 0 in blocks 13 and 14
2. Re-load interrupt processor from blocks 11 and 12
3. Continue monitor operation

d. Saving a File

1. Save interrupt processor in blocks 11 and 12
2. Re-load blocks 13 and 14 into pages 0 and 1 of field 0
3. Save whole program on tape
4. Save pages 0 and 1 in blocks 13 and 14
5. Re-load interrupt processor from blocks 11 and 12

e. Starting the System from the Console

1. Save pages 0 and 1 in blocks 13 and 14
2. Load interrupt processor from blocks 11 and 12
3. Continue

This type of system would also require a record to be kept of where the interrupt processor was at any particular time so that in the event of an error condition occurring within the monitor, the interrupt processor could be re-loaded if necessary.

7.8. Structure of the Command Decoder

An essential feature of a file handling system is that a user can manipulate files by using simple commands which can be entered into the system and executed immediately.

In view of the experience gained with FOCAL it was decided to develop the system using the techniques employed for the direct interpretation and execution of FOCAL commands. Input characters from the terminal device are therefore first loaded into a command input buffer until a carriage return character is received which signifies the entry of the command into the system. The command string is then decoded directly from the command input buffer, utilizing the sort and branch routines for linking to the subroutines associated with a particular command.

Errors in input command syntax and hardware faults when detected cause control to be transferred to an error recovery routine. This routine causes an error code to be printed out which can be used with a look up table to determine the nature of the error. Fig.45.

7.8.1. Command Structure

The basic command syntax has been developed in a form similar to that used in the OS8 for language compatibility purposes. The commands therefore have the following format:-

COMMAND DEVICE : FILENAME . EXTENSION .

COMMAND is one of the set of commands which have been
 included in the system described later.

DEVICE is the code name of the magnetic storage device
 upon which the required file, FILENAME . EXTENSION
 resides. As there are only two devices available,
 these being the two DECtape drives, the code names
 DTA0 has been assigned to drive unit 0
 DTA1 has been assigned to drive unit 1

FILENAME is between one and six alphanumeric characters long,
 starting with an alphabetic character

EXTENSION is a two character file identifier to denote the type
 of file being dealt with

<u>Error Code</u>	<u>Diagnostic</u>
0000	Restart from console
0174	Input buffer overflow
0200	CTRL/C
0262	Illegal command
0334	Select or write lock error on DECtape
0351	No FOCAL/SUPER-B codes on DECtape. Try another tape
0416	DECtape error during transfer of interrupt processor
0430	DECtape error during transfer of interrupt processor
0475	Read,write or select error on DECtape during block search
0511	Read,write or select error on DECtape whilst in check mode
0621	Storage filled by pushdown list
1650	Command input buffer is full
2222	No device has been specified
2224	No device has been specified
2250	Illegal device name
2301	Filenames cannot start with a "D"
2302	Filenames cannot start with a "D"
2303	Filenames cannot start with a "D"
2315	Too many characters in filename
2326	No "." after filename
2334	Filename extension must be two characters
2336	Filename extension must be two characters
2641	Not enough room left on DECtape
2661	Read,write or select error during SAVE command
2717	"," missing after BN extension in SAVE command
2722	Field should be a single character
2723	Field should be a single character
2732	"," missing between field and starting address
2734	Starting address should be in octal
2742	"," missing after starting address
2745	Number of blocks should lie in range 01 to 99
2746	Number of blocks should lie in range 01 to 99
2756	Number of blocks should lie in range 01 to 99
2757	Number of blocks should lie in range 01 to 99
2772	File specified in FIND command is not present
3001	File specified in ERASE command does not exist

Figure 45. Error Diagnostics for File Monitor System

<u>Error Code</u>	<u>Diagnostic</u>
3154	Read,write or select error on DECtape during ERASE or COPY
3160	Read,write or select error on DECtape during ERASE or COPY
3201	Attempt made to copy non-existent file
3206	Missing "<" in COPY command
3226	Filename already exists on specified device
3245	Not enough room left on DECtape
3270	Not enough room left in Directory
3405	Non-octal digit in field designation of RUN command
3420	Non-octal digit in starting address designation of RUN command
3475	Command must be terminated with RETURN
3504	":" missing after filename
3602	File not present for LOAD command
3610	Only .BN,.PR, or .DA file extensions are legal
3617	Too many blocks in .PR or .DA file
3633	PR file cannot be accommodated in FOCAL text buffer
3643	Read,write or select error on DECtape during LOAD command
3662	DA file cannot be accommodated in FOCAL variable area
3705	Filename already exists on specified device
3713	Directory is full
3727	Illegal file extension
3753	Read,write or select error on DECtape during SAVE command
3767	HELP command must be followed by file number (0 to 9)
3770	HELP command must be followed by file number (0 to 9)
4017	HELP file not present on DTAO
4231	Checksum error during WRITE or ACCESS command
4260	Field should be single octal digit in WRITE or ACCESS
4261	Field should be single octal digit in WRITE or ACCESS
4267	Only fields 0 or 1 are allowed in WRITE and ACCESS
4274	Incorrect character in starting address of WRITE or ACCESS
4301	Incorrect character in number of data transfers for WRITE or ACCESS
4375	"," missing between arguments of WRITE or ACCESS

Figure 45. Continued

The delimitting characters " " , " : " , " . " must be present within the command string as with FOCAL.

Internal subroutines used in the decoding and execution of the commands are given in Fig.46 which also lists the routines which are identical with those used with the FOCAL interpreter.

All internal subroutines have been created using the PAL III cross assembler system available, the list having been included on microfiche. Supporting documentation in the form of flowsheets is included in Appendix E, Fig.47 being the core configuration of the monitor system.

7.9. System Commands

The following section is concerned with the facilities which have been included in the system and the methods by which they have been implemented.

Examples of system use are also provided in Fig.48 which is one of the HELP files produced to provide users with system information.

7.9.1. DIRECTORY

This command has been designed to give a complete listing of all the file information available within the directory of a magnetic tape.

The output from this command is as shown in Fig.49 and it is achieved internally by using the "trace" facility which has been made available. This particular facility is the same as that used in FOCAL whereby two flip-flops are used. If both are zero when a character is reassembled from text, the character is printed at the console device. The file names are stored in the directory initially as stripped packed ASCII characters and have to be reformed using the internal routine GETC which uses the output text pointer to pick up the characters sequentially. With the trace facility enabled, each of these characters is reprinted after it has been reassembled.

The parameters associated with each file are from left to right (Fig.49)

1. File starting block, i.e. the position on the tape at which the file can be located. This number is printed out as a four digit decimal number using the internal binary to decimal decoding and printing routine PRNTLN.

2. The number of blocks on tape completely filled by this file. This is also printed out using PRNTLN and is therefore a 4 digit decimal number.
3. The number of words used by this file in the next block.
Four decimal digits using PRNTLN.
4. The address in core from which this file was saved.
It is a four digit octal address printed using PRNTOCT.
5. The field from which the file was saved. Four octal digit numbers printed using PRNTOCT.

7.9.2. SAVE Command

Facilities will be required for saving three types of files from core on to a magnetic storage device.

1. FOCAL Program Files
2. FOCAL Data Files
3. Executable binary files of either FOCAL system software or other binary software.

a. FOCAL Program Files

In the 8K version of FOCAL, programs are held in field 1 of memory in stripped packed ASCII form, lines in the program being linked together by the use of a pointer system.

There are therefore two possible ways in which files could be saved. Either the entire users text must be saved as an image file to be reloaded into the same place in core when required, or alternatively, use must be made of text output routines to load reformed ASCII characters into a buffer which may then be saved on tape. The reverse process would then be to read from tape into a buffer and use the text input routines to read from the buffer and repack the character in FOCAL program format.

This second method would provide a large degree of flexibility allowing users to save groups of lines or single lines or whole programs with the possibility of an overlay file system upon reloading. It was however decided that the organisation of such a system would require a large amount of modification to the FOCAL interpreter as well as the provision of a segmented buffer system for the transfer operations.

The first method was therefore chosen as being a compromise between flexibility and ease of implementation. By saving the whole of the

program text currently in the text buffer, the reloading of any program file will cause any previous information in the text buffer to be overwritten.

The PR extension was adopted for FOCAL program files and is used to denote that an area of core from 0100 in field 1 is to be saved as this is the starting point for FOCAL program text storage. The length of the file has to be determined by examination of the current end of text pointer in the FOCAL interpreter. The number of words required being converted into whole blocks and a single block containing the remaining number of words not accommodated within the integer number of blocks.

This data is used to update the directory of the particular device upon which the file is to reside.

b. FOCAL Data Files

Again, a procedure as described for FOCAL Program Files could have been adopted for the data file.

However, for reasons of simplicity, a system was adopted whereby the current data available in the symbol table is saved as a whole on the DECTape using a filename extension of DA.

The length of the file and position in core in which it resides is obtained by an examination of the FOCAL start and end of variables pointers. The file is saved and the directory is updated.

The versions of the SAVE command used for FOCAL program and data files are therefore .

SAVE DTA0 : FOCL GM . PR

SAVE DTA0 : FOCL GM . DA

and they cause automatic saving of program and data files.

c. Binary Files

It was assumed that any user who wanted to save binary files would first have written the executable binary program and therefore have a good working knowledge of computer operations.

It was therefore arranged that Binary files, whether FOCAL system software or other software would be saved as direct core image files to be reloaded back into core in the identical position. The saving of such a file would also require the user to specify the length of the file and the position in core from which it should be saved.

The format of the command for saving binary files is therefore:-

SAVE DTA1 : BINARY . BN , H , I , JK

where H is a single octal digit field setting

I is a one to four octal digit address from which the binary file is to be saved.

JK is a two decimal digit number, designating the number of whole blocks to be saved.

Binary files are saved as consecutive locations in core and can traverse across field boundaries. This allows a FOCAL system to be saved as a 64 block binary file.

d. Monitor System Saving

A facility has been included whereby the monitor system can be saved on DECtape. This was thought to be a necessary facility of the system as it allowed minor updates in system software to be accomplished by toggling in new instructions from the computer console. As a bootstrap loader is used (see a later section) this particular facility always ensures an up to date operating system.

The format of the command is simply

SAVE

and the system area of core, i.e. the whole of field 2, is automatically saved in blocks 20 through 52 of the DECtape in unit 0.

7.9.3. LOAD Command

All the files saved using the SAVE command are core image files which in general must be loaded back into the same position in core from which they were saved.

FOCAL Program files and binary core image files are therefore loaded into core from tape into exactly the same position.

With FOCAL Program files, a check must first be made to see whether the FOCAL text buffer is of sufficient length to accommodate the file. If not, an error code is issued. If the buffer is long enough, the file is loaded, and the end of text pointer in FOCAL adjusted accordingly.

With FOCAL Data files, it would not be convenient to load the file into the exact position from where it came. It must be loaded into the position of the current variable table.

Checks must first be made to determine whether sufficient space is available. When the file has been loaded, the end of variable pointer in FOCAL must be updated accordingly.

The versions of the load command are therefore:-

LOAD DTA0 : FOCLGM . PR

LOAD DTA1 : FOCLXM . DA

LOAD DTA1 : FCLVOI . BN

7.9.4. ERASE Command

An ERASE command is required as a housekeeping facility so that unwanted files can be deleted from the DECTapes and the space recovered for new files.

This could be done in either of two ways:-

1. By adopting a procedure of immediately recovering the space occupied by the deleted file, files below the deleted file being siphoned upwards and the directory modified accordingly. New files can then be saved automatically at the end of the current files on tape.
2. By adopting a procedure similar to that used in the OS8 system, whereby files deleted from the DECTape leave spaces which are then filled with new files when created.

This second procedure would require that when a new file is created, all of the empty spaces on the tape would have to be examined to determine which of the spaces was most suitable for the file in question. Empty spaces too small to accommodate files would be inevitably left, requiring the use of a special command to eventually recover any embedded inter-file spaces.

Quite sophisticated file saving procedures would have to be employed in order to perform these tasks.

It was therefore decided that the first method would be used thereby simplifying the procedure of file saving, but increasing the time required for file deletion because of the siphoning procedure required.

File siphoning has been accomplished by use of the area allocated to the directory buffer, ten blocks being read into core from the area occupied by the file and then written on to the DECTape into the new area. This cycle is continued until all the files have been moved and the space recovered.

The format of the ERASE command is

ERASE DTA1 : OBSLET . BN

7.9.5. RUN Command

As the file handling system has been developed essentially for the purpose of saving and recalling FOCAL programs, the save and load commands have been developed purely as file manipulation routines. After the use of these commands, control remains within the monitor system. There was therefore a need to provide a command to enable programs to be executed. In the case of FOCAL programs, this meant starting the FOCAL interpreter at address 0200 of field 0.

For other executable binary files created by experienced users, a command was required to start the binary program from any specified address within core.

A RUN command was therefore incorporated into the system so as to cater for the above requirements. The following formats are allowed:-

R

This format transfers control to 0200 field 0 and is the version of the command included specifically for the purpose of running the FOCAL interpreter.

In conjunction with this, an extension of the QUIT command has been provided in the FOCAL interpreter in order to return control back to 0200 field 2, the starting point of the monitor system. The QUIT command has the format

Q .

The other versions of the monitor RUN command are:-

R, A .. (1)

R, B, A . .. (2)

where A is a 1 to 4 octal digit starting address

B is a 1 to 4 octal digit field setting.

Version (1) assumes a field setting of 0 whereas version (2) provides the facility of totally defining a starting address.

7.9.6. COPY Command

The COPY command was included in the system in order to extend users file manipulation capabilities. Instead of files having to be loaded into core and then saved with existing commands, the COPY command utilizes the ten block directory buffer to make a copy of an existing file on the same DECTape or on another DECTape. If the copy is made on the same DECTape, a new name must be allocated to the file. If, however, a different DECTape is used, the same file name may be used. The system therefore allows important program files to be saved on a master tape, providing a safe copy in the unlikely event of system corruption.

The format of the command included is

COPY EXISTING FILE < NEW FILE

e.g. COPY DTA0 : FILE 1 . P R < DTA1 : FILE 1 . P R

7.9.7. FIND Command

A FIND command was included so that a DECTape directory could be examined to determine whether a named file was present, without the need to print out a lengthy directory of the file.

The command uses the internal directory search routine for the purpose. If the file named is not present on the selected device, an error code is printed on the terminal device. If the file is present, the system returns to await further commands, no error code being printed.

The format of the command is

F DEVICE : FILE : EX

7.9.8. ZERO Command

This command was included within the system mainly for initializing DECTapes to conform to a format specified for the system.

Control words have been included within the directory heading of the system so that DECTapes used for other file handling systems (OS8) cannot be read. The ZERO command is used to set up the control words at the beginning of the directory to clear the directory and to set up the first block from which files may be saved. Files are saved from block 70 onwards, the initial blocks on each tape being retained as a system area.

The form of the ZERO command is

Z DEVICE

This command will return with a ? and wait for further input from the user.

CTRL/S or CTRL/F will result in the device being zeroed, any other reply causing the command to abort, thereby affording some degree of protection for the inexperienced user.

7.10. Special Monitor Commands

7.10.1. HELP Command

In order to provide users with system information, it was thought to be necessary to include a command whereby users could obtain a brief guide to system operation. Alternatively, users could provide information about the running of their own files for other users.

This could have been accomplished by including an ASCII data handling facility within the monitor system, data being initially stored in a buffer, written as a file on to DECtape and then "replayed" when required.

An alternative procedure would be to allow direct running of a FOCAL program from the monitor system. The information file could then be created in FOCAL and the whole of the FOCAL interpreter and program saved as a binary file. With suitable extensions to the monitor system, the binary file could then be executed from a single command within the monitor.

The second method although requiring the saving of the FOCAL interpreter for each HELP file produced and hence a slightly inefficient usage of magnetic storage, would provide a simple and flexible compromise by utilizing facilities already available with MONITOR system.

HELP files are therefore created by loading FOCAL and creating an information file as a FOCAL program. The whole of the FOCAL interpreter and program text is saved as a 64 block binary file on device DTA0 using the SAVE command. The file names for such files are HELP n . BN where n is a single digit in the range 0 to 9.

The binary file can then be executed using the command

HELP n

where n again defines the HELP file number.

Automatic loading and execution was accomplished by use of the

internal routines of the monitor.

Fig.48 shows one of the HELP files which has been created to provide users with information about system operation and commands.

7.10.2. Interprocessor Communication

The development of an interprocessor buffer, using M series I/O modules available for the PDP 8E and the buffered accumulator facilities available in the PDP8, made it possible for accumulator to accumulator transfer to take place between the two machines.

Unfortunately the system requires the accumulators of both machines to be available for transfers (non direct memory access), programmed data transfers would therefore hold up the operation of an existing program, even if the system was driven under interrupt.

As the PDP 8E has to act as a controller for the DECTape transfers, synchronous data transfers from tape to PDP8 via the PDP 8E would be virtually impossible because the PDP 8E has no external clock. This would make it difficult to incorporate the PDP 8E/DECTape system into a dual processor Real-Time FOCAL operating system. The use of the machines in this fashion would require the development of extremely sophisticated software.

It was therefore thought that they could be best used as a storage medium for all FOCAL software used on the PDP8 and the PDP 8E. Such a system would require handlers for the interprocessor buffer to be resident in both machines for transfers in both directions.

As the PDP8 has only a minimum amount of free core space available, the PDP 8E file monitor system was chosen as the controller for the proposed system, using a slave loader program in the PDP8. As magnetic storage would effectively replace the need to use paper tape storage, the slave program in the PDP8 could be made to occupy the area of core normally associated with the Binary Loader.

As the PDP 8E cannot control data transfers between itself and the PDP8 concurrently with DECTape transfers, a buffering system would be required.

This could be achieved either by using a small buffer, transferring perhaps ten blocks of data per time or else by using the whole of the lower

8K of core in the PDP 8E and creating a duplicate core image in the 8E of programs in the 8.

This latter method was selected because of the ease with which it could be implemented. Routines already available within the monitor system are used to save the core image on the DECtape once the transfer process had been completed, the reverse process being achieved by reloading the program into core and transferring the core image back again.

The disadvantage of such a system would be that any program resident in the lower 8K of the 8E would be overwritten and require reloading. This would however produce little inconvenience as all system software would be reloadable from the magnetic tape using the monitor system.

Thus it was decided to implement core image transfers using programmed data transfers rather than interrupt driven data transfers which would have required significant modifications to the interrupt processor.

7.10.2.1. WRITE and ACCESS Commands

The WRITE command causes a core image transfer from the PDP 8E to the PDP8 and has the following format:-

WRITE , X , Y , ZZZZ

where X is the field of transfer only fields 0 and 1 being allowed

Y is the address within the field, from which the transfer is to start

ZZZZ is a 1 to 4 octal digit number used to define the number of consecutive locations to be transferred.

The maximum size of data transfer which can take place at any one time is 4K of memory, achieved by setting ZZZZ to 0.

For transfers in the reverse direction, the ACCESS command is used.

ACCESS , X , Y , ZZZZ

Both of these commands must be used in conjunction with a Special Loader written for the PDP8

Each time a data transfer is requested by using either the WRITE or ACCESS commands, the PDP 8E sends four initial data words to the PDP8 via the interprocessor buffer, the field, the starting address, the number of words and a switch for the direction of transfer. During the transfer, a two by 12 bit checksum is calculated in both machines. These are checked

against each other at the end of the transfer process by sending both of the twelve bit words computed in the PDP8 to the PDP 8E. The words are then compared with the two words computed in the 8E ; non-correspondence produces an error code at the monitor terminal device.

The special loader in the PDP8 resides in the area of core normally occupied by the binary loader, i.e. page 37, field 1. It may be initialized by starting from the computer console in which core control remains within the loader after the data transfer.

An alternative procedure to facilitate the use of the interprocessor buffer from FOCAL, an extension has been made to the FOCAL ERASE command to transfer control to the loader program.

Thus the FOCAL user may type

E E

at his terminal and control will be transferred to the loader.

After the data transfer has been completed, control is returned immediately to the FOCAL interpreter.

Fig.50 shows an example of the use of the WRITE and ACCESS commands for loading the FOCAL system and program stored on DECtape into the PDP8 and also the methods by which FOCAL programs running on the PDP8 may be saved.

7.11. Bootstrapping the System

The system was developed using a cross assembler in the ICL system 450 computer (parts in the OS8 operating system using Symbolic Editor + PAL8) and therefore exists initially as a paper tape binary program. It is therefore necessary to load this program with the binary loader and then save it on the DECtape in the system area automatically. This has been done and in the initialisation of the system, the system area is saved in blocks 20 through to 52 of the DECtape on unit 0. The interrupt processor is also immediately written into blocks 11 and 12 of the DECtape on unit 0.

As the system is present as a file in a DECtape, it would seem sensible if the system could be loaded from the tape directly once the system has been built rather than having to use paper tape versions. The set of instructions shown in Fig.52 is the bootstrap loader for the system, and is used to load in a much larger bootstrap loader, stored in block 0 of the tape

on unit 0. This maxi bootstrap is then used to load in the system from tape, providing data transfer error detection by use of checksums. This maxi bootstrap is saved in block 0 when the system is initially loaded from paper tape.

Fig. 51 shows the configuration adopted on DECtapes for the allocated system area.

7.12. Program Chaining Facilities provided by the system

The methods which have been employed in the development of the Monitor system and also the FOCAL configuration used (see Chapter 4) allow stored FOCAL program modules to be chained together.

For example, if a lengthy program is required, it may be split up into a data initialisation section, computation section and an output section in the following manner:-

# L DTAO:PRDAIN.PR	Load initialisation program
# R	
?00.00	FOCAL entered
*G	
	} Input of data
*Q .	Return to monitor
? 0000	
# L DTAO:PRCOMP.PR	Load computation program
# R	
?00.00	FOCAL entered
*G	
	} Data analysis program
*Q .	Return to monitor
?0000	
# L DTAO:PROPUT.PR	Load output program
# R	
? 00.00	FOCAL entered
*G	
	} Output of data

Similarly data tables could be created by using an initialisation program, and then stored on DECtape for use at some later stage, e.g.:-

#L DTAO:PRDAIN.PR	Load data input program
#R	Run program
? 00.00	FOCAL entered
*G	} Enter data for first file
*Q .	Return to monitor
? 0000	
#S DTAO:DATF2.DA	Save data file
#R	Re-run for next file
? 00.00	
*G	} Enter data for second file
*Q .	Return to monitor
? 0000	
#S DTAO:DATF3.DA	Save data file
#	

This particular feature although somewhat cumbersome and not quite as convenient as a direct program chaining facility implemented via calls within FOCAL does, in fact, make the system particularly easy to use. Its simplicity ensures that all users know exactly what is happening and can easily control operations for the selection of the correct program module.

7.13. Discussion

The system developed has been used extensively by Undergraduates, Postgraduates and members of staff within the School of Chemical Engineering.

As yet, no major defects have been encountered when using the system and the time required to become conversant with system operation is minimal. It would also appear that most of the aims and objectives set out prior to the development of the system have been achieved.

One of the major advantages of the system is the ease with which extensions can be provided to the system. Minor modifications can be corrected by inserting in the new sections from the computer console, the system then saved using the special version of the SAVE command.

<u>Internal Routine</u>	<u>Calling Sequence</u>	<u>Function</u>
DIRRED	DRCTRY	Read or write the directory of a specified device into or out of the 10 block directory buffer. The accumulator and link must be set prior to calling the routine. AC=0, unit 0; AC=4000, unit 1; LINK=0, READ; LINK=1, WRITE.
RW1314		Used for reading from or writing to blocks 13 and 14 of DECTape on unit 0 in order to restore or save pages 0 and 1 of field 0
RW1112		Used for reading from or writing to blocks 11 and 12 of DECTape on unit 0 in order to restore or save the interrupt processor of the File Monitor System
RSTRT	RESWOP	Uses the routines RW1112 and RW1314 to save pages 0 and 1 of field 0 in blocks 13 and 14 of DECTape on unit 0 and to restore the interrupt processor from blocks 11 and 12
TERMN8	XTRMN8	Uses the routines RW1112 and RW1314 to save the interrupt processor in blocks 11 and 12 of DECTape on unit 0 and to restore pages 0 and 1 from blocks 13 and 14
SEARCH		Searches the tape of a given unit for the block number which is held in the accumulator

Figure 46. Internal Routines used by the File Monitor System

<u>Internal Routine</u>	<u>Calling Sequence</u>	<u>Function</u>
TCHECK	CHECK	Tests contents of core against contents of DECTape for the last tape transfer which occurred
XRTL6	RTL6	As in FOCAL
XTESTC	TESTC	As in FOCAL
XSORTC	SORTC	As in FOCAL
XPUSHA	PUSHA	As in FOCAL
XPUSHJ	PUSHJ	As in FOCAL
PD2	PUSHF	As in FOCAL
PD3	POPF	As in FOCAL
INPUT		As in FOCAL
SORTB	SORTJ	As in FOCAL
TD8E	DTAPE	Used for controlling DECTape transfers. Parameters are set up prior to entering this routine. First return from this routine indicates that an error has occurred during the transfer.
UTRA	GETC	As in FOCAL
CHIN	READC	As in FOCAL
OUT	PRINTC	As in FOCAL
XPRNT	PRNTLN	Used to decode a 12 bit binary word, held either in the accumulator or in UNITS, into a 4 digit decimal number and then prints out the digits in the correct order.

Figure 46. Continued

<u>Internal Routine</u>	<u>Calling Sequence</u>	<u>Function</u>
PACBUF	PACKC	As in FOCAL
ETWO	ERROR	As in FOCAL with a few minor modifications
RUB1		As in FOCAL
XSPNOR	SPNOR	As in FOCAL
XTESTN	TESTN	As in FOCAL
XPOPJ	POPJ	As in FOCAL
FILESTR	FILSTR	Picks up file parameters from the directory list and uses them to set up the parameters for the DECTape handling routine
DVCE	DEVICE	Used for picking up the device specified by the input text. If no device is specified in the text, unit 0 is automatically selected, otherwise DTA0 selects unit 0 and DTA1 selects unit 1. The codes DTA0 and DTA1 are used to form a hash code which is then used in a sort and branch routine
GETFIL	PUSHJ GETFIL	Used for obtaining the filename and extension from the input text and storing it in the buffer FLBUFR. The buffer is 4 words long and characters are packed two per word. If a filename is less than six characters in length, the remaining characters in the buffer are set to the code for a space. Correct command terminators are tested for and return is made using a POPJ

Figure 46. Continued

<u>Internal Routine</u>	<u>Calling Sequence</u>	<u>Function</u>
GETFW2		Picks up two characters from input text for storing in FLBUFR prior to a file search. 1st return is taken if a single character and a terminator is found, 2nd return is taken if two characters are found
XSRCH	FLSRCH	Searches the directory of a DECtape for a file whose name is held in FLBUFR. If the file is not found, XRT5 will point to next available space in directory buffer, XRT4 will point to FLBUFR and the counter CNTR will be set to -4, the first exit will be taken. If the file is found, XRT5 will point to the file parameters, XRT4 will point to nothing, the counter will be zero and the second exit will be taken
OCTPRNT	PRNTOCT	Prints a number held either in the accumulator or in UNITS as a 4 digit octal number
SA	JMS I 0132	Reads in a 1 to 4 octal digit address and stores it in NUMBER
TCARTN	TESTCR	Tests a character for a carriage return. If not found an error exit is taken

Figure 46. Continued

<u>Internal Routine</u>	<u>Calling Sequence</u>	<u>Function</u>
DVCFLE	FIIEDVC	Reads the required device from text and tests syntax. The filename is picked up from the text and the directory of the required device read into the directory buffer. The directory is then searched for the specified filename. The first exit is taken if the file is not found in the directory. The second exit is taken if the file is found in the directory.

Figure 46. Continued

Super D Core Map Field 2

Zero Page Pointers									
0	Command Decoder					Directory Read/Write			
1						Free			
2	Interrupt Processor Shifting Routines					Rtl6, Testc and Sortc			
3	Pushdown List Controls					Free			
4									
5	Debounce Control								
6	Routine								
7									
10	Getc					Printc			
11	Packc					Error Recovery Routine			
12	Rubout					Free			
13	Device								
14	Smor					Comlst, Comgo etc			
15	Getdvc					Free			
16	Flsrch					Getfil			
17	Directory					Zéro			
18									
19	Save								
20	Erase					Inouts			
21						Seto			
22	Copy					Inout			
23	Sa					Octornt			
24	Tester					Free			
25	Load					Save			
26									
27	Help								
28									
29	Write and Access Commands					Free			
30									
31	Free					Command Input Buffer			
32									
33									
34									
35									
36									
37									
38									
39									
40									
41									
42									
43									
44									
45									
46									
47									
48									
49									
50									
51									
52									
53									
54									
55									
56									
57									
58									
59									
60									
61									
62									
63									
64									
65									
66									
67									
68									
69									
70									
71									
72									
73									
74									
75									
76									
77									
78									
79									
80									
81									
82									
83									
84									
85									
86									
87									
88									
89									
90									
91									
92									
93									
94									
95									
96									
97									
98									
99									

Figure 47.

YOU HAVE ORVIOUSLY FOUND OUT HOW TO USE THE HELP COMMAND

CONTROL CHARACTERS

SHIFT/O DELETES LINE UP TO LEFT HAND MARGIN
 LINEFEED RE TYPES THE CURRENT LINE IN SUPERR BUFFER
 DELETE ERASES LAST CHARACTER IN LINE.ECHOES \ FOR EACH DEPRESSION
 OF THE DELETE KEY
 RETURN ENTERS THE COMMAND STRING TO BE PROCESSED
 CTRL/C RETURNS TO COMMAND MODE WITH ERROR CODE ? 0200

SUPERR COMMANDS

DIRECTORY

ALLOWABLE FORMS

D GIVES DIRECTORY OF UNIT 0
 D DT00
 D DT01 GIVES DIRECTORY OF UNIT 1

THE DIRECTORY LISTING IS IN THE FOLLOWING FORM

FILE.EX START BLOCK BLOCKS PART BLOCKS START ADDRESS FIELD
 E.G
 #D

TDR00T.EN	0070	0001	0000	7300	0000
RIMPUN.EN	0071	0001	0000	7400	0000
JFOCAL.EN	0072	0064	0000	0000	0000
LUNAR .PR	0136	0008	0029	0100	0001
TTT0E .EN	0145	0032	0000	0000	0000

Figure 48. HELP File Provided by File Monitor System

LOAD

ALLOWABLE FORMS

L DTAC:FILE.EX LOADS NAMED FILE FROM UNIT 0
 L :FILE.EX ..
 L DTAI:FILE.EX LOADS NAMED FILE FROM UNIT 1

EXAMPLE

TO LOAD FOCAL FROM SUPERD
 #L DTAI:JFOCAL.PN
 #

ERASE

ALLOWABLE FORMS

E DTAC:FILE.EX ERASES NAMED FILE FROM UNIT 0
 E :FILE.EX ..
 E DTAI:FILE.EX ERASES NAMED FILE FROM UNIT 1

THE ERASE COMMAND ALSO MOVES FILES BELOW THE NAMED FILE SO THAT ALL
 NEW FILES CAN BE ADDED TO THE BOTTOM OF THE LIST .THUS FLIMINATING THE
 NEED TO FILL IN EMPTY SPACES IN THE DIRECTORY LATER

COPY

ALLOWABLE FORMS

C DEVICE:FILE1.EX<DEVICE:FILE2.EX

THE FIRST FILE IS THE FILE TO BE COPIED .THE SECOND FILEIS THE NEW FILE
 COPIES CAN BE MADE ON TO THE SAME DEVICESAS LONG AS THE NEW FILE HAS A
 DIFFERENT NAME TO THE OLD FILE

Figure 48. Continued

```

RUN
ALLOWABLE FORMS
R Y,XXXX    RUN PROGRAM FROM FIELD Y ADDRESS XXXX
R XXXX      RUN A PROGRAM FROM FIELD 0 ADDRESS XXXX
R           RUN A PROGRAM FROM FIELD 0 ADDRESS 0200

X AND Y MUST BE OCTAL DIGITS
XXXX CAN BE 1,2,3 OR 4 OCTAL DIGITS

EXAMPLE TO LOAD AND RUN FOCAL
#L DTAI:JFOCAL.RN
#R 0,200
OR
R 200
OR
R ? 00.00
*
FOCAL IS NOW READY TO RECEIVE COMMANDS
TO RETURN TO SUPER D MONITOR FROM FOCAL
*G .
? 0000
#
THE SUPER D MONITOR IS NOW WAITING TO RECEIVE COMMANDS
-----

```

Figure 48. Continued

ZERO

ALLOWABLE FORMS

Z DTAO WILL ZERO DECTAPE UNIT 0
 Z ..
 Z DTAI WILL ZERO DECTAPE UNIT 1

ON ENTERING THE COMMAND INTO THE SYSTEM THE TTY WILL ECHO

?

AND WAIT FOR A REPLY FROM THE USER

THE USER NOW HAS THREE CHOICES AVAILABLE TO HIM

1. TYPE CTRL/S .WHICH WILL WRITE SYSTEM.PN ON THE DIRECTORY HEADING(TO BE USED FOR THE SYSTEM DEVICE)
2. TYPE CTRL/F .WHICH WILL WRITE SPACES ON THE DIRECTORY HEADING
3. ANY OTHER CHARACTER WILL APOPT THE ZERO COMMAND WITHOUT DOING ANYTHING

NOTE. USE THE ZERO COMMAND WITH CARE AS VALUABLE FILES MAY BE LOST

SAVE

ALLOWABLE FORMS

S DTAO:FILE.PR SAVES A FOCAL PROG ON UNIT 0
 S :FILE.PR ..
 S DTAO:FILE.DA SAVES A FOCAL SYMPOLE TABLE ON UNIT 0
 S :FILE.DA ..
 S DTAO:FILE.PN,A,P,C SAVES A BINARY FILE ON UNIT 0
 S :FILE.PN,A,P,C ..
 S DTAI:FILE.PR SAVES A FOCAL PROG ON UNIT 1
 S DTAI:FILE.DA SAVES A FOCAL SYMPOLE TABLE ON UNIT 1
 S DTAI:FILE.PN,A,P,C SAVES A BINARY FILE ON UNIT 1
 S SAVES THE SYSTEM AREA ON UNIT 0 BLOCKS 20 TO 51

A IS THE FIELD IN WHICH THE AREA TO BE SAVED RESIDES

IT SHOULD BE A SINGLE OCTAL DIGIT

P IS THE STARTING ADDRESS OF THE AREA TO BE SAVED

1,2,3 OR 4 OCTAL DIGITS

C IS THE NUMBER OF BLOCKS TO BE SAVED

IT MUST BE TWO DECIMAL DIGITS IF FROM 01 TO 99

Figure 48. Continued

FILE NAMES

FILE NAMES CAN BE ONE TO SIX CHARACTERS IN LENGTH BEGINING WITH
AN ALPHABETIC CHARACTER OTHER THAN D

FILE EXTENSIONS

THERE ARE ONLY THREE ALLOWABLE FILE EXTENSIONS

PR FOR A FOCAL PROGRAM
DA FOR A FOCAL SYMPOLE TABLE
BN FOR A CORE IMAGE BINARY FILE

DEVICES

DTA0 FOR DECTAPE UNIT 0
DTA1 FOR DECTAPE UNIT 1

GENERAL INFORMATION

1. SUPER D RESIDES IN FIELD 2 OF A 12K SYSTEM
2. ITS STARTING ADDRESS IS 0200
3. ITS INTERRUPT PROCESSOR RESIDES IN THE FIRST TWO PAGES OF FIELD 0
4. THE INTERRUPT PROCESSOR IS KEPT ON DECTAPE UNIT 0 IN BLOCKS 11 AND 2
AND IS SWAPPED IN AND OUT OF CORE WHEN REQUIRED

Figure 48. Continued

ERROR CODES

0173	INPUT BUFFER OVERFLOW
0262	NO SUCH COMMAND
0334	SELECT OR WRITE LOCK ON DECTAPE
0351	TAPE NOT ZEROED
0416	SELECT OR WRITE LOCK ON DECTAPE
0430	..
0475	..
0511	..
0621	STORE FILLED BY PDL
1650	COMMAND BUFFER FULL
2222	NO DEVICE SPECIFIED
2224	..
2250	NO SUCH DEVICE
2301	FILES MUST START WITH A TO Z BUT NOT D
2302	..
2303	..
2315	TOO MANY CHARACTERS IN NAME
2326	NO . AFTER NAME
2334	TWO CHARACTERS IN EXTENSION PLEASE
2336	..
2641	TAPE IS FULL
2661	SELECT OR WRITE LOCK ON TAPE
2717	MUST HAVE , AFTER FILENAME WHEN SAVING RN FILE
2722	FIELD SHOULD BE SINGLE NUMBER
2723	..
2732	NO COMMA AFTER FIELD
2734	NONE OCTAL DIGIT IN INPUT
2742	NO COMMA AFTER START ADDRESS
2745	BLOCKS 01 TO 99
2746	..
2756	..
2757	..

Figure 48. Continued


```

3001 NO FILE
3154 DECTAPE ERROR
3160 ..
3201 NO FILE
3206 NO < FOR SEPERATION
3226 FILE EXISTS
3245 TAPE FULL
3270 DIRECTORY FULL
3405 NON OCTAL DIGIT
3420 ..
3475 RETURN NOT FOUND
3504 : NOT FOUND AFTER DEVICF
3602 NO FILE
3610 ILLEGAL FILE EXTENSION
3617 TOO MANY BLOCKS INPR OR DA FILE
3633 FILE TOO BIG FOR FOCAL
3643 DECTAPE ERROR
3662 FILE TOO BIG FOR FOCAL
3705 FILE ALREADY EXISTS
3713 FULL DIRECTORY
3727 ILLEGAL FILE EXTENSION
3753 DECTAPE ERROR
4017 HELP FILE NOT AVAILABLE
5043 TAPE ERROR RELOAD PAPER TAPE VERSION
0000 CONSOLE RESTART
0200 CTRL/C
*
```

Figure 48. Continued

D DTAI

```

PRESS .PR 0070 0006 0007 0016 0100 0001
PRES .PR 0077 0007 0096 0100 0001
PROPS1.PR 0085 0007 0118 0100 0001
KVALUE.PR 0093 0004 0090 0100 0001
KCONST.PR 0098 0004 0037 0100 0001
FLASH .PR 0103 0009 0111 0100 0001
GIKCOL.PR 0113 0008 0112 0100 0001
PRSEW.PR 0122 0003 0036 0100 0001
TMPDEW.PR 0126 0004 0032 0100 0001
PRSEBUB.PR 0131 0003 0039 0100 0001
TMPBUB.PR 0135 0004 0039 0100 0001
HEX1 .PR 0140 0001 0079 0100 0001
LUNAR .PR 0142 0008 0029 0100 0001
RIMPUN.BN 0151 0001 0000 7400 0000
HELP4 .BN 0152 0064 0000 0000 0000
OS8BWT.BN 0216 0001 0000 7300 0000
JDUO .BN 0217 0064 0000 0000 0000
OCTDMP.BN 0281 0002 0000 1600 0000
TTT0E .BN 0283 0032 0000 0000 0000
LOADRS.BN 0315 0001 0000 7600 0001
BUFOUT.BN 0316 0004 0000 0000 0000
LABEL .BN 0320 0003 0000 0200 0000
SPOOF .PR 0323 0007 0074 0100 0001
MTDUP8.BN 0331 0005 0000 0000 0000
RTDU08.BN 0336 0064 0000 0000 0000
RTF8DN.BN 0400 0064 0000 0000 0000
RTF8LG.BN 0464 0064 0000 0000 0000

```

Figure 49. Example of the DIRECTORY Command of the File Monitor System

<u>PDP-8E</u>	<u>ACTION</u>	<u>PDP-8</u>
#LOAD :FCL8EN.BN	Load FOCAL into core of the PDP-8E from DTAO.	
#LOAD DTA1:FCLLVN.PR	Load required FOCAL program into PDP-8E.	Run the Special Loader from the PDP-8 console
#W,0,0,0	Transfer the whole of field 0 from the PDP-8E to the PDP-8.	
#W,1,0,7577	Transfer field 1 from the PDP-8E to the PDP-8 ensuring that the Special Loader is not overwritten.	Start the FOCAL interpreter from address 0200 field 0 700.00
#A,0,0,0	Link to the Special Loader from FOCAL on the PDP-8 and transfer the data to the PDP-8E using the ACCESS command.	*E E
#A,1,0,7577	FOCAL will return to command mode after each field transfer. Re-link to the Special Loader and transfer next field.	*E E
#SAVE DTA1:FCLLVN.PR	Save the FOCAL program on magnetic tape.	*
#		

Figure 50. Example of the Use of the Inter-processor Buffer

BLOCKSCONTENTS

0	System Bootstrap Loader
1 to 10	Directory of files stored on the DECtape
11 and 12	Interrupt Processor for File Monitor System
13 and 14	Pages 0 and 1 of field 0
15 to 19	Not used
20 to 51	Saved core image file of File Monitor System
52 to 69	Not used
70 to end	Saved files

Figure 51. Block Allocation of Formatted DECtapes

ADDRESS OCTAL	CONTENTS OCTAL	SYMBOLIC	SOURCE LINE	
7300	1312	BEGIN,	TAD SEARCH	//START DECTAPE IN REVERSE
7301	4312		JMS SEARCH	//BACK TO END ZONE
7302	4312		JMS SEARCH	//MARK TRACK CODE OF 31
7303	6773	QUAD,	SDSQ	//WAIT FOR QUAD LINE FLAG
7304	5303		JMP .-1	
7305	6777		SDRD	//READ DATA
7306	3726		DCA I POINT	//STORE DATA
7307	2326		ISZ POINT	//MOVE ON TO NEXT
7310	5303		JMP QUAD	//MORE TO COME YET
7311	5732		JMP I START	//NOW TO MAXI-BOOTSTRAP
7312	2000	SEARCH,	2000	
7313	1300		TAD BEGIN	//SET FOR DECTAPE 0
7314	6774		SDLC	//LOAD COMMAND REGISTER
7315	6771	INLOOP,	SDSS	//WAIT FOR SINGLE LINE
7316	5315		JMP .-1	
7317	6776		SDRC	//READ COMMAND REGISTER
7320	0331		AND C77	//ISOLATE MARK TRACK BITS
7321	1327	BKFND,	TAD M22	//END ZONE INITIALLY
7322	7640		SZA CLA	//FOUND YET
7323	5315		JMP INLOOP	//NO KEEP LOOKING
7324	2321		ISZ BKFND	//RESET FOR MT CODE 31
7325	5712		JMP I SEARCH	//RETURN
7326	7574	POINT,	7574	//LOAD ADDRESS
7327	7756	M22,	-22	//END ZONE CODE
7330	7747	M31,	-31	//START CODE
7331	0077	C77,	0077	//MARK TRACK MASK
7332	7617	START,	7617	//START ADDRESS

Figure 52 . Bootstrap Loader for File Monitor System

CHAPTER 8.

Conclusions and Recommendations

8.1. Introduction

One of the major problems encountered when using computers is that of communication. The computer only "understands" binary words stored within its memory and the general user only "understands" language statements. In order to make it possible for the user to communicate meaningfully with the computer, some form of "translator" must be interposed between them.

The "translator" could take the form of an experienced programmer, capable of creating programs at machine code level and developing all applications programs for users. This type of system generally results in the potential user having to wait a considerable time before his program becomes operational and the final product is inclined to be more what the programmer thinks the user requires, rather than what the user actually requires. Further difficulties arise when the program requires modification, either to correct it or to make it more flexible, as the experienced programmer must be involved once again.

A more flexible approach to this communications problem would be to have some form of software or hardware "translator" to convert users' language statements into binary machine words. This would allow users to create their own applications software and modify it as necessary without having to involve a third party at any stage.

For off-line data processing problems, this latter method has been employed for a long time in the form of compiler-based or interpreter-based high level language systems. However, for on-line control problems, the former method has been used in the past because it has been necessary to make more efficient use of core, peripheral equipment and processing time.

Recently however, there has been a trend towards developing and using more flexible systems in on-line situations. This has resulted in "Fill-in-the-Form" techniques, hardware programming aids and high level language systems being adopted for real time applications. The type of system employed is dependant upon the nature of the real time application and the facilities required by the users of the system.

8.2. Choice of Operating System

In an Education and Research environment, off-line facilities are required for data processing, mathematical modelling and simulation studies. In addition to this, on-line facilities are required for the control of experiments, data acquisition from experiments and data processing of the results of experiments. Rather than having different systems and languages for on-line and off-line applications, it would be more efficient if the same or similar systems were adopted in both cases.

Programming systems based upon the use of either "fill-in-the-form" techniques or hardware pushbutton programming techniques were discounted because their specific orientation towards plant control meant that they lacked the facilities for general data processing problems.

The choices available were therefore either to select an existing high level language system and modify it to provide on-line facilities or alternatively to develop a new programming system. As any system developed would have had to include general data processing facilities, it was decided that under the circumstances it would be better to take an existing high level language system and use it as the basis for an on-line system.

Although FORTRAN is a "standard" high level language, it is generally a compiler-based language. The use of a compiler-based language on a small minicomputer, without mass storage, is extremely tedious, involving the user in several steps of creating, compiling and running using paper tape as an intermediate storage medium for source, object and compiled programs.

FOCAL, an interpretive language, was therefore selected as being a suitable basis for an operating system. The core resident nature of the interpreter avoids the necessity for loading separate sections of the operating system during the various stages of program development and execution. The interpreter also possesses the advantages of single statement or group execution and extensive text editing facilities, thereby making the process of program testing and correction easy and relatively fast. Its major disadvantage stems from the fact that program execution speeds are much slower than for a similar compiled program.

8.3. Real-Time FOCAL Systems

The FOCAL interpreter was first extended to provide on-line facilities for a single user. The tests described in Appendix A show that the system is fully capable of dealing with a control and data acquisition problem typical of those which would be encountered within a laboratory environment. As expected, the system was quite slow in terms of program execution, being capable of a maximum data sampling rate of the order of 20 per second. This however was not thought to be too slow to prevent the system being extended so as to permit two users to time share its facilities.

Although most of the slowness of the FOCAL systems may be attributed to the method of operation of the interpretive system, the architecture of the computer on which it is being used also influences operating speeds.

8.3.1. Hardware Limitations

The 12-bit word length of the PDP-8 computer only allows direct addressing of 128 words of memory and indirect addressing of 4096 words of memory. As most references to variables will be outside the 128 word range, indirect addressing has to be used. Thus the time taken to access a word of information will be greater than that for computers with larger word lengths.

Allied to this is the fact that the PDP-8 has only a single accumulator and a simple instruction set which requires that all data transfers and manipulations occur through the accumulator. The time taken for even simple operations is therefore greater in the PDP-8 than in machines with multiple accumulators and more sophisticated instruction sets.

For example:-

PDP-8 (12 bit word)

PDP-11 (16 bit word)

CLA CLL	MOV A,B)
TAD I A	reference to A) 3 words
DCA I B	reference to B)
.	.
.	.
.	.
.	.
A, X	A: 023456
B, Y	B: 000000
.	
.	
.	
X, 2345	
Y, 0000	
Execution time 7.5 μ secs	Execution time 4 μ secs
Words used 7 (84 bits)	Words used 5 (80 bits)

All multiplications and divisions whether unsigned integer or floating point must be done by software on a minimal PDP-8 system. This will obviously have a detrimental effect upon program execution speeds. For example, a software routine to perform unsigned integer multiplication of two twelve bit numbers will take approximately 500 μ secs on the PDP-8. On machines with this function incorporated into the instruction set, the process would take of the order of 10 μ secs. Similarly with floating point software, a single operation whether addition, multiplication, subtraction or division will take about a milli-second. The same process done with parallel processing floating point hardware takes about 10 μ secs.

If a "push down list" is required on the PDP-8, all management functions must be done by software, unlike other machines which have hardware controlled stack management. This must also add to the execution times experienced in FOCAL particularly within the expression evaluation routine, where numerous stack operations are required.

Within the PDP-8, all interrupts are connected to a common interrupt bus which means that an interrupt will have the same effect as any other interrupt. Thus in order to detect which peripheral device has caused the interrupt, a software skip chain must be used to test the status of the flags of all devices. This obviously is less efficient in terms of processing time usage than for a system which has priority vectored interrupts, which immediately link the interrupting device with its handling software. This may not have a great effect in a single user system but it is essential for efficient use of peripheral devices in a multi-access system.

8.3.2. Software Limitations and possible enhancements.

Most of the time required by the FOCAL interpreter for program execution can be attributed to the variable search and line location routines used. If the execution times of FOCAL programs were to be reduced, these routines would have to be modified. The major problem would then be to modify them so that the flexibility of the system was not reduced significantly.

The same type of line structure could be maintained within a system which "semi-compiled" source lines on input. System subroutines for commands could be linked directly into a line before it was stored within the FOCAL text area. Thus instead of having to decode command lines during execution, a subroutine address could be inserted at entry time which would cause a direct jump at execution time. The resultant savings in execution times could be of the order of 0.5 milliseconds but in order to maintain the same editing facilities, more sophistication would have to be added to line editing routines. A two level command structure would also have to be implemented, as it would not be necessary to semi-compile command lines executed in immediate mode, or commands like MODIFY and WRITE which are generally used in direct mode.

Methods of by-passing line location routines by a similar process would be extremely difficult to implement as branches to lines not existing at the time of source line input could not be changed to an address until the referenced line was inserted. This would mean that a process similar to compilation would have to take place prior to the program being run. Many of

the better characteristics of the interpretive system would be destroyed by such modifications thereby defeating the object of using the interpreter.

A similar problem exists within the variable search routines. If variable addresses were to be inserted instead of the variable name, a compilation stage would have to be inserted between the creation and running of the program.

A method of reducing variable search times would be to employ a slightly different data storage structure. Instead of allowing all variables to have subscripts, they could be split into arrayed variables and non-arrayed variables in a manner similar to that employed in FORTRAN. The name of an arrayed variable could then be used to locate the position of the stored data vector, and the subscript used to determine the position of the defined variable within the vector. This would of course require the use of array dimensioning before program execution and would create a two section variable table for arrayed variables and non-arrayed variables, with a resultant slight decrease in the time taken to access a variable.

All these modifications have one common drawback in that they need memory space for their inclusion in the system. In a minimal 8K system, the extra core would have to be found at the expense of variable and text storage areas.

Finding sufficient core space in order to add functions to FOCAL for real-time computing was a constant problem. Many of the facilities had to be implemented by using multiple argument function calls requiring one or two arguments in order to define a sub-function of a function call. As each function evaluation takes of the order of 1 millisecond, the time taken to execute a particular function is lengthened. This situation could be avoided if more core was available allowing a separate function to be included for each group of peripherals. Most of the functions could in fact be included as system commands, thereby avoiding the initial excursion through the function evaluation routine which all FOCAL functions must take. It would also avoid having to use an extra command in order to invoke the function call. In this manner, time savings of about 2 milliseconds per function could be obtained.

8.4. FOCAL as an Off-line Programming Language

The development of a system for using DECtapes as a medium for saving FOCAL programs and system software was undertaken in order to extend the computation facilities so that FOCAL could be used extensively as an off-line system as well as an on-line system.

It might be argued that the OS8 operating system already available, provided more than adequate off-line programming facilities. However, as has been previously stated, the OS8 system is necessarily complex because of the comprehensive file handling facilities which it provided. The user not only has to learn the programming language but also has to learn the commands of a number of utility programs in order to create source programs as files and manipulate them.

The version of FORTRAN which is supported by OS8 will allow quite large programs to be run on a mini-computer system when the overlay facility is fully implemented. Its major disadvantage is that even small programs take of the order of ten minutes to be compiled into an executable core image program when using DECtape as the mass storage device. This process can be very tedious during the program development stage where large numbers of errors have to be found and corrected.

The main objective was therefore to provide FOCAL with simple but comprehensive file handling capabilities in the form of a monitor system.

The monitor structure was adopted in order to provide the system with added flexibility. If the file manipulation facilities had been implemented through direct commands from FOCAL, then each different version of FOCAL would have had to have been saved on separate tapes which would have required loading each time a different version was required. A monitor structure however allowed different versions of FOCAL to be saved as core images which could be loaded from commands within the monitor. This has the added advantage of allowing other binary core images of programs or data to be handled, a facility useful to users wishing to program in assembly code.

The system allows for a fairly primitive overlay structure which must be controlled directly by the user. Programs have to be developed in a modular form, each section being loaded and executed after the termination of a previous section. Although this appears to be rather crude, it allows

the user to make decisions at the end of each program module. In cases where it is extremely difficult to program complex logic, this can be advantageous since it minimises the possibility of taking wrong program paths.

8.5. Use of Interpretive Programming Systems

The selection of any operating system is essentially constrained by the type of computer available and the applications for which it is to be used.

If, for example, most of the programs to be developed are likely to be "standard modules" to be used frequently without change, then a system is required in which program execution speeds are rapid. The development time in such a situation is only of minor importance. Alternatively, in an environment where programs are developed for a specific application, to be used only a few times, then development time should be made as short as possible. Similarly, where there are several users all wishing to use the system for their own data acquisition purposes, development times should again be of prime importance.

In order to make program development times as short as possible, it is essential to provide a programming language with a simple but comprehensive syntax and a system which is easy to operate. FOCAL and other interpretive languages do in fact provide these requirements. Subsequent experience with FORTRAN on an RSXII/PDP-II real time operating system has shown that it can take considerably longer for users to become familiar with the compiler based system and that program development times are also far greater.

Hardware checking by direct line execution is another advantage which the interpreter has. In compiler based systems, if any hardware checking is done, then short programs must be written so as to test the item.

8.6. Conclusions

The investigation described in this thesis demonstrates that the use of an interpreter based system is feasible for the programming, by the user, of real-time applications. The advantages associated with the flexibility of an interpreter are shown to outweigh the disadvantages associated with its limited speed of execution. Whilst some improvements in execution

times could be achieved by a restructuring of the interpreter, significant improvements would require additional hardware resources.

REFERENCES

1. PERONE, S.P. and JONES, D.O. "Digital Computers in Scientific Instrumentation". McGraw-Hill Book Company, 1973.
2. "Computer Software for Process Control". Report of the working party on Real Time Computer Software, May-December, 1968. Plant Engineering and Energy Division, B.I.S.R.A.
3. "Future Trends in the Minicomputer Industry". Proceedings Minicomputers in Instrumentation and Control, June 1973 p.117. (Paker, Cain and Morse).
4. KIPINAK, W., QUINT, P. "Assembly vs Compiler Languages". Control Engineering Vol.15, February 1968. pp.93-98.
5. LEE, W.T., BOARDMAN, R.M., HIGHAM, J.D. "Block Diagrammatic Programming in Computer Control Projects". 2nd U.K.A.C. Convention on Advances in Computer Control. I.E.E. Conference Publication No. 29, 1967. Paper C.6.
6. LEE, W.T. "Experience with an On-Line Conversational Control Software System". A symposium on experiences with software in computer control applications. Institute of Measurement and Control, July 1969. pp.12-18.
7. HIGHAM, J.D., JONES, R.E. "Conrad Software in the Paper and Cement Industries". *ibid* pp.19-25
8. HIGHAM, J.D. "Conrad III - Conversational On-Line Software for DDC Supervisory and Sequence Control". Conference on Computer Aided Design, Southampton, April 1969.
9. BAILEY, S.J. "Push Button Programming for On-Line Control". Control Engineering, Vol.15, August 1968, pp.76-79.
10. EWING, R.W., GLAHN, G.L., LARKINS, R.P., ZARTMAN, W.N. "Generalised Process Control Programming Systems". Chemical Engineering Progress, Vol.6, No.1. January 1967, pp. 104-110.
11. MARKHAM, G.W. "Fill in the Form Programming". Control Engineering, Vol.15, May 1968. pp.87-91.
12. SMITH, C.L. "Fill in the Forms Computer Languages for Process Control". Chemical Engineering, March 3rd, 1975. pp.151-156.
13. KELLY, V.H., WAKEFIELD, A.J. "APE - A New Approach to Programming for On-Line Control". 2nd U.K.A.C. Conference on Advances in Computer Control, Paper C.11.

14. CLOUGH, J.E. "Fortran for On-Line Control". Control Engineering Vol.15, March 1968., pp.77-81.
15. GASPAR, T.G., DOBROHOTOFF, V.V. "New Process Language Uses English Terms". Control Engineering, Vol.15, October 1968, pp. 118-121.
16. TURNER, G., NORMAN, D.E., BELCH, R.F. "A Computer Operating System for Plant Control". 2nd U.K.A.C. Convention on Advances in Computer Control. Paper C.8.
17. MECKLENBURGH, J.C., MAY, P.A. "PROTRAN, A Fortran Based Computer Language for Process Control". Automatica, Vol.6., 1970. pp.565-579.
18. GERTLER, J. "High Level Language for Process Control". The Computer Journal, Vol.13, No.1., February 1970. pp.70-75.
19. NEVE, N.J.F. "Coral 66 - The U.K. National and Military Standard". Trends in On-Line Computer Control Systems. I.E.E. Conference Publication, No. 127, pp.139-146.
20. EYRE, D.M., WILLIAMS, H.B. "The Application of Coral 66 to Control Computers". Software for Control. I.E.E. Conference Publication, No. 102, p.155.
21. WETHERALL, P.R. "Coral 66 - A language for the control of Real Time Processes". A symposium on experiences with software in computer control applications. Institute of Measurement and Control, July 1969., pp.56-59.
22. BARNES, J.P. "Process Control Systems using RTL/2". Software for Control. I.E.E. Conference Publication, No.102, p.75.
23. BURGESS, A.M. "Users experience with a Real Time Language RTL/2" *ibid*
24. CROWLEY-MILLING, M.C. "Interpretive Software for a Large One-Off Process". *ibid* p.63.
25. CROWLEY-MILLING, M.C., HYMAN, J.T., SHERING, J.C. "The Multi-Computer Control System for the CERN 400 GEV Accelerator". Trends in On-Line Computer Control Systems, I.E.E. Conference Publication, No.127, pp.101-107.
26. DUNCAN, J.B. "Anatomy of PC²; Process Control Language". Control Engineering, April 1974, pp.42-44.
27. WILKINS, C.L., KLOPFENSTEIN, C.E. "Laboratory Computing with Real-Time Basic". Chemtech, November 1972. pp.681-686.

28. ANDERSON, R.E. "Programming Languages for Laboratory Control"
Journal of Chromatographic Science, Vol.7, 1969, pp.725-730.
29. PIKE, H.E.Jr. "Process Control Software". Proceedings I.E.E.
Vol.58, January 1970, pp.87-97.
30. SPANG, H.A. "The Structure and Comparison of Three Real-Time
Operating Systems for Process Control". Automatica,
Vol.8, 1972, pp.49-64.
31. PURSER, W.F.C., JENNINGS, W.M. "The Design of a Real-Time
Operating System for a Minicomputer". Software - Practice
and Experience, Vol.5., 1975, pp.147-167.
32. GERTLER, J., SEDLAK, J. "Software for Process Control - A Survey"
Automatica, Vol.11, 1975, pp.613-625.
33. SECREST, D. "Time Sharing Experimental Control on a Small
Computer". Industrial and Engineering Chemistry, Vol.60,
1968, No.6., pp.74-80.
34. Small Computer Handbook 1968. Digital Equipment Corporation.
35. SALMON, J.D. "A Versatile Computer Control System".
Instrument Engineer, October 1966, pp.110-118.
36. SALMON, J.D. "Computer Compatible Hardware and Software for
Process Control". Advances in Computer Control, I.E.E.
Conference Publication, No.29, Paper C.19.
37. BECK, M.S., WAINWRIGHT, N. "DDC of Chemical Processes".
Control, Vol.12, January 1968, pp.53-57.
38. BOOTH, A.D. "Resolution of (Complex) Gas Chromatograms using
a Digital Computer". Transactions of the Society of
Instrument Technology, Vol.19, No.1., March 1967, pp.12-16.
39. BAUMANN, F., HERLICKA, E., BROWN, A.C., BLESCH, J.
"Quantitative Evaluations of Chromatograms by Digital
Computers". Journal of Chromatographic Science, Vol.7.,
November 1969, pp.680-684.
40. THOMPSON, J.W. M.Sc. Dissertation, University of Bath, 1972.
41. Programming Languages, 1970, Vol.2, Digital Equipment Corporation.
42. KONGAS, M. "Focal - An Easy Way to Real Time". Decus Europe
Proceedings, 1972, pp.223-224.
43. ENGLISH, J.C. "Focal used in Data Acquisition and Control Systems -
Advantages and Disadvantages". Decus Proceedings,
Spring 1972, pp.191-194.

44. SIEGEL, W., WHITTLE, K. "Using FOCAL in Research", *ibid* 195-202
45. WREGE, D.E., HARMER, D.S. "FOCL/F.. An Extended Version of 8K FOCAL/69". *ibid* pp.213-219
46. CAVE, P.R. "Real Time Fortran in a Mechanical Engineering Laboratory". Decus Europe Proceedings. pp.70-80.
47. Advanced FOCAL Technical Specifications. Digital Equipment Corporation.
48. Programming Languages 1969 Vol. 2. Digital Equipment Corporation.
49. WEEKS, C.H. "An Extended Version of FOCAL for Multi-User Data Logging and Control". Decus Australia Proceedings, 1972.
50. Small Computer Handbook, 1973. Digital Equipment Corporation.
51. CHARLESWORTH, A.S. "Assembler Program Documentation". Decuscope, Vol.14, No.1., 1975., pp.8-12.
52. PAYNE, W.D. "Speeding up FOCAL" Decuscope. Vol.10, No.13, 1971, pp.79.
53. CHARLESWORTH, A.S. Private Communication.
54. CHARLESWORTH, A.S. "Altmode/Echo". Decuscope, Vol.12, No.3., 1973. p.17.
55. EAST, L.V. "An Improved Random Number Generator for FOCAL 9/15". Decoscope, Vol.11., No.2., 1972.
56. PRINGLE, R.G.C. "PAL III Symbolic Assembler User's Guide". University of Bath Computer Unit (December, 1970).
57. PDP-8 User's Handbook. Digital Equipment Corporation
58. LOCKETT, A.D. To be published.
59. SHINSKEY, F.G. "Feedforward Control Applied". I.S.A. Journal, November, 1973.
60. ROVIRA, A.F., MURRILL, P.W., SMITH, C.L. "Modified Algorithm for Digital Control".
61. GOFF, K.W. "A systematic Approach to DDC Design". I.S.A. Journal, Vol.13, No.12., December 1966. pp.44-54.
62. DUNN, C. Final Year Project Report. Department of Chemical Engineering, University of Bath.

63. ROBINSON, K.S. Ph.D. Thesis, University of Bath.
64. STAGEMAN, M. Final Year Project Report Department of Chemical Engineering, University of Bath.
65. WILLS, D.M. "Tuning Maps for Three Mode Controllers". Control Engineering, April 1962. pp.104-108.
66. LOPEZ, A.M., MILLER, J.A., SMITH, C.L., MORRILL, P.W. "Tuning Controllers with Error Integral Criteria". Instrument Technology, November 1967. pp.57-62.
67. EDLER, J., NIKIFORUK, P.N., TINKER, E.B. "A Comparison of the Performance of Techniques for Direct On-Line Optimization". The Canadian Journal of Chemical Engineering, Vol.48, August 1970, pp.432-440.
68. BEVERIDGE, G.S.G., SCHECHTER, R.S. "Optimization: Theory and Practice". McGraw-Hill, 1970.
69. HEAPS, H.S., WELLS, R.V. "The Effects of Noise on Process Optimization". The Canadian Journal of Chemical Engineering, December 1968. pp.319-324.
70. DANNENBERG, K.D., MELSA, J.L. "Stability Analysis of Randomly Sampled Digital Control Signals". Automatica, Vol.11., pp.99-103.
71. AKAIKE, H. "Effect of Timing Error on the Power Spectrum of Sampled Data". Annals Institute of Statistical Mathematics, Vol.11., 1960, pp.145-165.
72. BLACKMAN, R.B., TUKEY, J.W. "The Measurement of Power Spectra from the point of view of the Communications Engineer". Dover, New York, 1959.
73. SWINNERTON-DYER, H.P.F. "The Calculation of Power Spectra". Computer Journal, Vol.5., 1962, pp.16-23.

APPENDIX A.

Use of Real-Time FOCAL

A.1 Introduction

During the development of the extensions to FOCAL to provide real-time programming capabilities, it was possible to test most of the input and output routines for logical errors by using the direct mode of operation available within the interpretive language. However, for the purpose of testing the system as a whole, for timing errors, control algorithm operation, ease of use and other defects, it was necessary to select a simple but effective experiment for indirect operation.

A.2. Equipment and Experimental Procedure

The equipment used is illustrated in Fig. A.1. The heat exchanger/cooling tower equipment is generally used for undergraduate teaching purposes.

The flow transducer FT1 was of the propeller type providing a pulse train, counted and displayed locally in terms of pulses per second. The output of the transmitter was also supplied as an input to one of the input counter cards of the computer interface.

The two temperature transducers TT1 and TT2 were resistance thermometers, the output being provided as a voltage signal for a bridge network and displayed locally. The voltage levels were also supplied as input to the analogue to digital converter of the computer.

Both the steam flow rate and the water flow rate were controlled with automatic diaphragm valves, V1 and V2 respectively, operated by manostat pressure regulating devices. The manostat device maintains a constant air pressure on the diaphragm valve until a request for a change in position is received from the pulse width modulation output system of the computer. The pressure is then altered in order to change the valve position.

This equipment had been used during the initial trials of FDYN in order to test the effectiveness of the lead/lag network as a multipurpose control algorithm. It had also been used to test the logic of the PCI algorithm used in the FCON function. This previous experience with the system had shown that the flow measurement was particularly "noisy"

due to quantization errors and disturbances in flow pattern. Also the control of flow was subject to noise arising from appreciable stiction and hysteresis in the pneumatic diaphragm operated control valve.

It was therefore decided that an appropriate test of the system software would be an on-line determination of the optimum parameters in the PCI algorithm when used to control the flow rate in the heat exchanger. As the system was noisy, the effectiveness of various on-line search techniques could be evaluated in the presence of process noise.

A.3. Choice of Objective Functions

Any technique used for the optimization of controller settings requires the definition of a performance criterion. A simple measure of performance for feedback systems would be to compare the magnitude and duration of errors produced by a known disturbance. This can be achieved by integrating a suitable function of error with respect to time.^{65,66} The following integral error performance criteria have been widely used in controller optimization studies:-

1. Integral of Absolute Error
2. Integral of Squared Error
3. Integral of Time and Absolute Error

The difference between these criteria lies in the weighting given to the magnitude or the duration of errors produced.

It was decided that a set point disturbance would be the easiest method to implement with the equipment available. Initial experiments showed that of the three above criteria, the IAE was the least susceptible to the noise on the input signal and that a simple trapezoidal integration of error values was as effective and reproducible as any other simple method of numerical integration. It was also found that the direction of the step change had little effect upon the magnitude of the integral obtained.

It was therefore decided that the optimum control algorithm parameter for the flow control loop should be determined by using the IAE as an objective function in conjunction with trapezoidal error integration for a step change in set point of 35% full scale to 70% full scale. A loop

scanning rate of 1 second³⁷ was adopted and proved to be adequate.

A.4. Lattice Search Procedure

Before proceeding to a fully automatic direct search procedure it was thought necessary to first examine the response surface of the system. This entailed the evaluation of the IAE for various values of controller gain and integral action time so as to produce a contour map of the response surface in the region of the optimum.

This lattice search procedure is in effect a pattern search procedure (see later) requiring the evaluation of the objective function at various regular lattice points on the response surface of the control loop. It involved numerous function evaluation but proved to be very useful in the latter stages of this project when the approach of the on-line search procedure towards the optimum could be observed.

Figs. A2 and A3 are the program and associated flowsheets developed to perform the lattice search. At each selected lattice point, a series of experiments were performed, evaluating the IAE for the particular step change in set point in both upwards and downwards directions. When the required number of experiments had been completed at the chosen lattice point, the mean and standard deviations were computed and logged for both directions of set point change. This procedure was then repeated at various other points over the region of the response surface surrounding the optimum so that a contour map could be produced.

Fig.A4 gives a brief description of the operation of the program.

A.5. Results

Figs.A5 to A7 show the results obtained for an upward change in set point as plots of proportional band. The standard deviations at the various lattice points are also expressed on these diagrams and serve the purpose of showing how the IAE value was affected by the noise on the input signal.

The proportional band (PB) in the result was related to the control loop gain factor in the FCON function

$$S\ Z = FCON (L, SP, IT, SC, KG, CO)$$

$$\text{by } KG = \frac{100}{PB}$$

As the loop scan rate had been fixed at 1 second, the integral action time was expressed in seconds.

Fig.A8 shows the results expressed as a contour map of IAE values for variation in integral action time and proportional band. This particular figure showed that the optimum was located within a long narrow valley which could make it difficult to locate with an on-line search procedure, particularly when the system was also subject to noise. However it would provide a suitable test in order to determine any limitation in the computer operating system and to provide experience as to the most efficient ways of using the system.

A.6. On-Line Search Procedures

On-line search techniques can be divided into three main categories:-

1. Gradient search techniques in which the selection of the next point is determined on the basis of the local gradients of the response surface in the region of interest.
2. Pattern search techniques in which the selection of the next point is based upon some pre-determined pattern. Lattice searches belong to this class of techniques.
3. Random search techniques in which the selection of the next point is random within the given area of interest.

EDLER et al⁶⁷ suggested that in the presence of process noise, random search techniques were more effective than pattern or gradient search techniques and were particularly easy to implement.

Gradient search techniques described in reference 67 were only effective in the situation of little or no process noise. They also required the evaluation of partial derivatives at each new point, thereby increasing the computational load.

For processes with a limited number of controlled input variables a "Simplex" pattern search technique was suggested to be effective even with appreciable noise. They also had the added advantage of being easily implemented, requiring only the evaluation of one new point per cycle as in the random search.

Having already determined the nature of the response surface in the region of the optimum and observed that the main problems were noise and dimensionality rather than ridges and multiple optima, it was decided that implementation of a "Simplex" search and a random search would provide a suitable test of the system software.

A.6.1. Simplex Search Procedure ⁶⁸

A Simplex search procedure is a pattern search which takes a regular geometric figure as its basis. In the case of two independent variables, this geometric figure is an equilateral triangle. Experiments are performed such that the objective function is evaluated at the points formed by the vertices of the geometric figure. After the initial Simplex has been generated, the search proceeds in the following manner:-

1. So as to maintain the geometric figure, the new vertex is selected as a reflected point in the opposite side of the figure.
2. A vertex is rejected if it produces an inferior value of the objective function when compared with the value of the objective function at the other vertices.
3. No return can be made to a point which has just been rejected. This particular rule is necessary so as to avoid the possibility of the Simplex oscillating between two vertices. In these circumstances the second worst vertex is rejected.
4. If a vertex remains unchanged for several experiments, the step size of the Simplex is reduced and the search continued. The search is terminated when the step size has been reduced to an extent whereby

the optimum has been located to the required degree of accuracy.

A regular geometric figure can only be obtained if changes in the independent variables produce similar changes in the value of the objective function. Therefore when starting the Simplex search, it is necessary to examine the local gradients in the region of the starting point in order to fix the dimensions of the Simplex and effectively scale the independent variables.

For a two dimensional search, 68, recommends the vertices shown in Fig.A.17.

The new vertex can be obtained in the following manner assuming that $(x, y,)$ is the rejected point and (x_4, y_4) will be the new vertex.

$$x_4 = x_3 + x_2 - x_1$$

$$y_4 = y_3 + y_2 - y_1$$

thereby producing the vertex as a reflected point in the opposite side.

Figs. 9 - 10 show the program and associated flowsheets developed to perform the Simplex search procedure. At each selected vertex, a single experiment was performed to determine the IAE for a step change of 35% to 70% full scale. Fig.A.11 gives a brief description of the operation of the program.

A.6.1.1. Results

Fig.A.12 shows the results obtained from two such simplex searches, plotted on the contour map determined by the lattice search procedure. Both the searches shown eventually terminated within the area found to be the optimum for the objective function and step change chosen. These two sets of results are particularly interesting as they show the effect of the process noise upon the system, both searches being started under identical conditions. Search 1. was initially sent in the wrong direction and took a long time to return to the correct direction and eventually locate the optimum value.

Other searches performed from different starting points with different step lengths also terminated within the region of the optimum but like the two searches illustrated, took various directions and numbers of function evaluations to do so.

A.6.2. Random Search Procedure

A simple random search procedure of the form described in reference 69 was implemented. From a starting point of (x_i, y_i) the objective function is determined at points $(x_i + dx R_x, y_i + dy R_y)$ where dx and dy are maximum step lengths allowed, and R_x and R_y are random numbers in the range $-1 < R < +1$. Wherever a point (x_{i+1}, y_{i+1}) is reached at which the response is better, (in this particular case a smaller value of the IAE) than the response at the point (x_i, y_i) , the point (x_{i+1}, y_{i+1}) is taken as the new starting point of the search.

The search proceeds until no better response is located within a predetermined number of attempts. At such a point, the objective function is redetermined and the search continues for another cycle. This added condition avoids the possibility of obtaining a false minima due to noises.

Figs. A13 and A14 show the program and associated flowsheets developed to perform the search procedure. Fig.A15 gives a brief description of the program.

A.6.2.1. Results

Fig.A16 shows the initial stages of two random searches plotted upon the contour map determined by the lattice search procedure. Both the illustrated searches eventually terminated within the region of the optimum as did other searches performed from different starting points with different step sizes.

A.6.3. Conclusions

With regard to the search procedures, the limited number of experiments performed seemed to confirm the suggestion that the random search procedure was as effective, if not more so, than the pattern search method of the simplex procedure.

Both methods adopted were easily implemented although the random search techniques required slightly less effort than did the Simplex search technique.

With regard to the operation of the Real-Time FOCAL system, no adverse performance was detected during the experiments which were performed.

The timing loop structure adopted in the development of the Real-Time FOCAL operating system coped adequately with the computational loads involved. In simple tests run during the optimization experiments no skipping of sampling time was detected. Admittedly, scan rates of 1 second were being used for most of the time but these were adequate for the system chosen.

The FCON control function and associated interrupt driven output system also proved satisfactory in operation.

The debugging and editing facilities available within the FOCAL language proved to be most useful, particularly in program development, where various parameter and control logic needed to be changed and enhanced frequently.

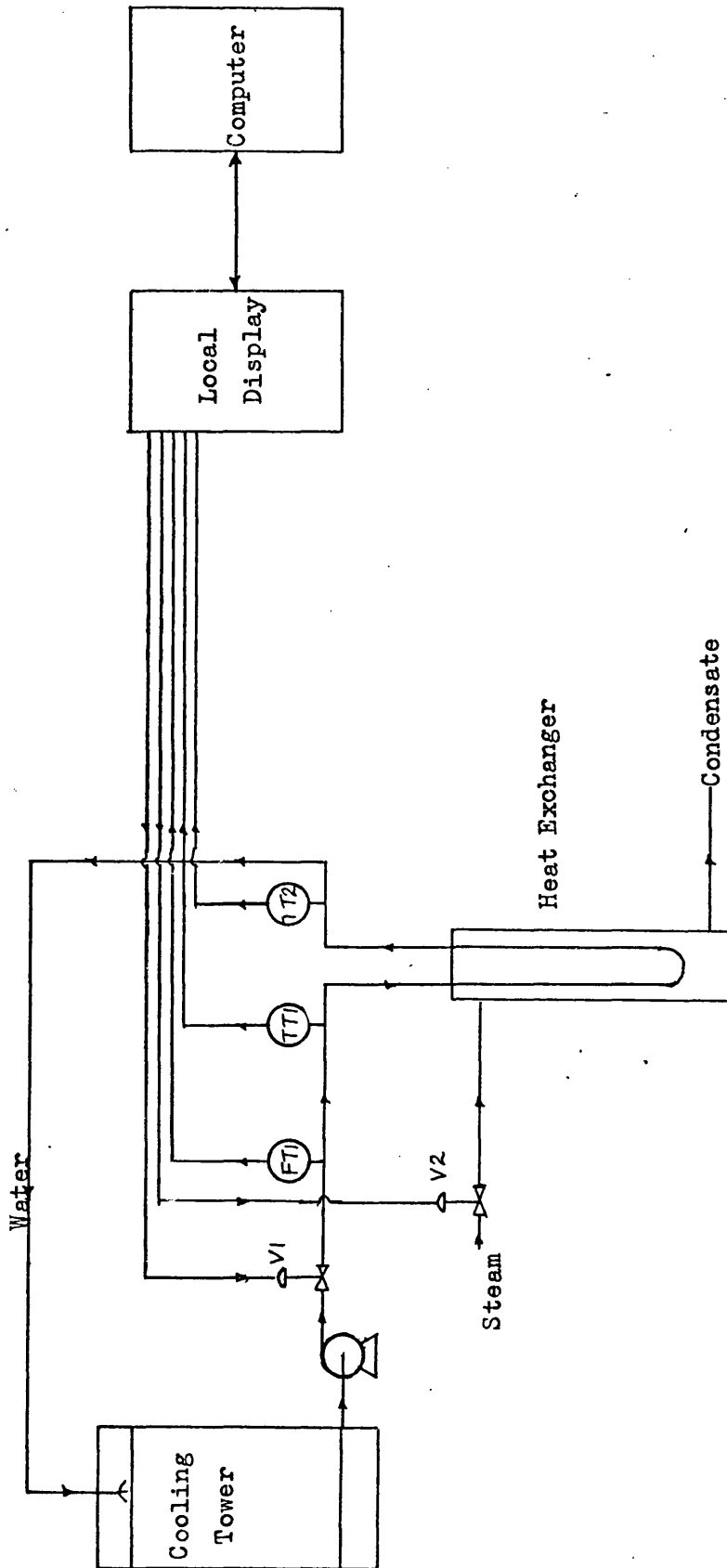


Figure A1. Schematic Diagram of Equipment used for On-line Experiments

C-8A SYMBOLS FOCAL @1974 NEW TTY

```

01.01 A "INPUT CODE",IC,"INPUT CHANNEL ADDRESS",CI,!
01.03 A "LOOP NUMBER",L,"OUTPUT CHANNEL ADDRESS"CO,!
01.05 A "INTEGRAL ACTION TIME",IT,"PROPORTIONAL BAND"PB,!
01.06 S IT(15)=17;S PP=0
01.07 A "DEAD ZONE PULSES"DZ,"SCAN INTERVAL"SC,"INT DEAD ZONE"DZ(2),!
01.08 A "NUMBER OF EXPTS EVEN PLEASE",TT,"LINE OUT ",LO,!
01.09 A "INITIAL SET POINT",IS,"FINAL SET POINT"LS,!
01.11 S KC=100/PB;S SP=IS
01.12 S TM(10)=FTIM(0SCS)+60*(FTIM(0MNS)+60*(FTIM(0HRS)+24*FTIM(0DYS)))
01.16 S NM=1;S UT=0;S DE=0;S VR(1)=0;S VR(2)=0;S CF=-1;S J=0;S CT=0
01.20 I (FLAG(SF(1)))1.25
01.23 G 1.20
01.25 D 6.0
01.27 I (CF)1.40,1.43,1.35
01.30 S SP=LS;S CF=1
01.31 T !
01.32 G 1.20
01.35 D 3.0
01.37 G 1.20
01.40 S TM=FTIM(0SCS)+60*(FTIM(0MNS)+60*(FTIM(0HRS)+24*FTIM(0DYS)))
01.42 I (TM-TM(10)-10)1.20,1.60,1.60
01.43 I (X-SP)1.20,1.44,1.20
01.44 S TM(10)=TM
01.46 I (SP-IS)1.50,1.30,1.50
01.50 S SP=IS;S CF=1
01.52 T " "
01.54 G 1.20
01.60 S CF=0
01.62 G 1.43

02.01 F WE=1,2,NM-2;S VR(1)=VR(I)+(SM(WE-1+I)-UT(I))*2
02.03 S VR(I)=FSQT(VR(I)*2/TT)
02.05 R

```

Figure A2. FOCAL Program for Lattice Search Procedure

```

03.01 S J=J+1;S F=FARS(X-SP)
03.03 I (1-J)3.09
03.05 S SM(NM)=F;R
03.09 I (80-J)3.30
03.11 I (D2(2)-E)3.20
03.13 S E=0;S CT=CT+1
03.15 I (LO-CT)3.40
03.17 G 3.22
03.20 S CT=0
03.22 S SM(NM)=SM(NM)+2*E;R
03.30 T "UNSTABLE OR TOO SLOW",!,"PROP BAND"PR,"IAT"IT,!
03.32 S GF=-1;S J=0;S CT=0
03.34 G 3.66
03.40 S SM(NM)=(SM(NM)+E)*SC/2
03.42 T %5.3,SM(NM),J-1
03.44 S NM=NM+1;S GF=-1;S J=0;S CT=0
03.46 I (TT-NM)3.50
03.47 S TM(10)=FTIM(0SCS)+60*(FTIM(0MNS))+60*(FTIM(0HRS))+24*FTIM(0DYS))
03.48 R
03.50 F WE=1,2,NM-2;S UT(1)=UT(1)+SM(WE);S UT(2)=UT(2)+SM(WE+1)
03.52 S UT(1)=UT(1)*2/TT;S UT(2)=UT(2)*2/TT
03.54 F I=1,1,2;D 2.0
03.58 T !,"PROP BAND"PR,"IAT"IT,"GAIN"KG,!
03.60 T "      UP      DOWN",!
03.62 T "      MEAN      STD DEV      MEAN      STD DEV",!
03.64 T %5.3 UT(1),VR(1),"      ",UT(2),VR(2),!!!
03.66 D 12.0
03.68 S NM=1;S UT(1)=0;S UT(2)=0;S VR(1)=0;S VR(2)=0
03.70 R

```

Figure A2. continued

```

06.01 S X=FIN(IC,C1)
06.02 I (FABS(X-SP)-DZ)6.03,6.03,6.04
06.03 S X=SP
06.04 S Z=FCON(L,X,SP,IT,SC,KG,CO)
06.07 R

08.01 S SF(1)=10;S SF(2)=300;S SF(3)=50;S N=10
08.03 S Z=VLAC(OST,N,SF(1),SF(2),SF(3))
08.05 Q

12.01 S PP=PP+1
12.03 I (6-PP)12.10
12.05 S IT=IT+.1
12.07 C 12.11
12.10 S IT=IT(15);S PP=0;S PB=PB-5;S KG=100/PR
12.11 S TM(10)=FTIM(OMNS)+60*(FTIM(OMNS)+24*FTIM(ODYS))
12.12 R
*
```

Figure A2. continued

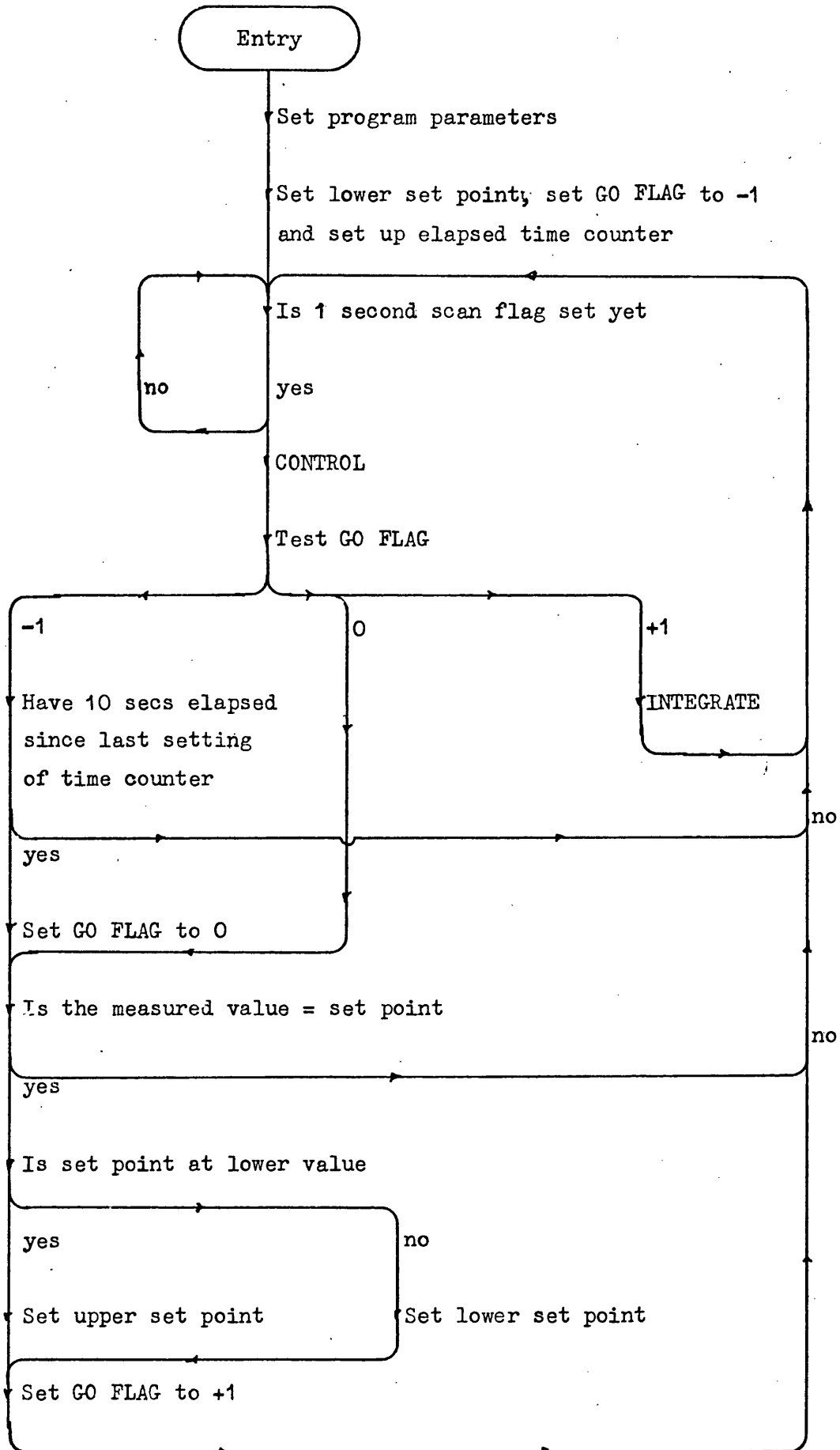
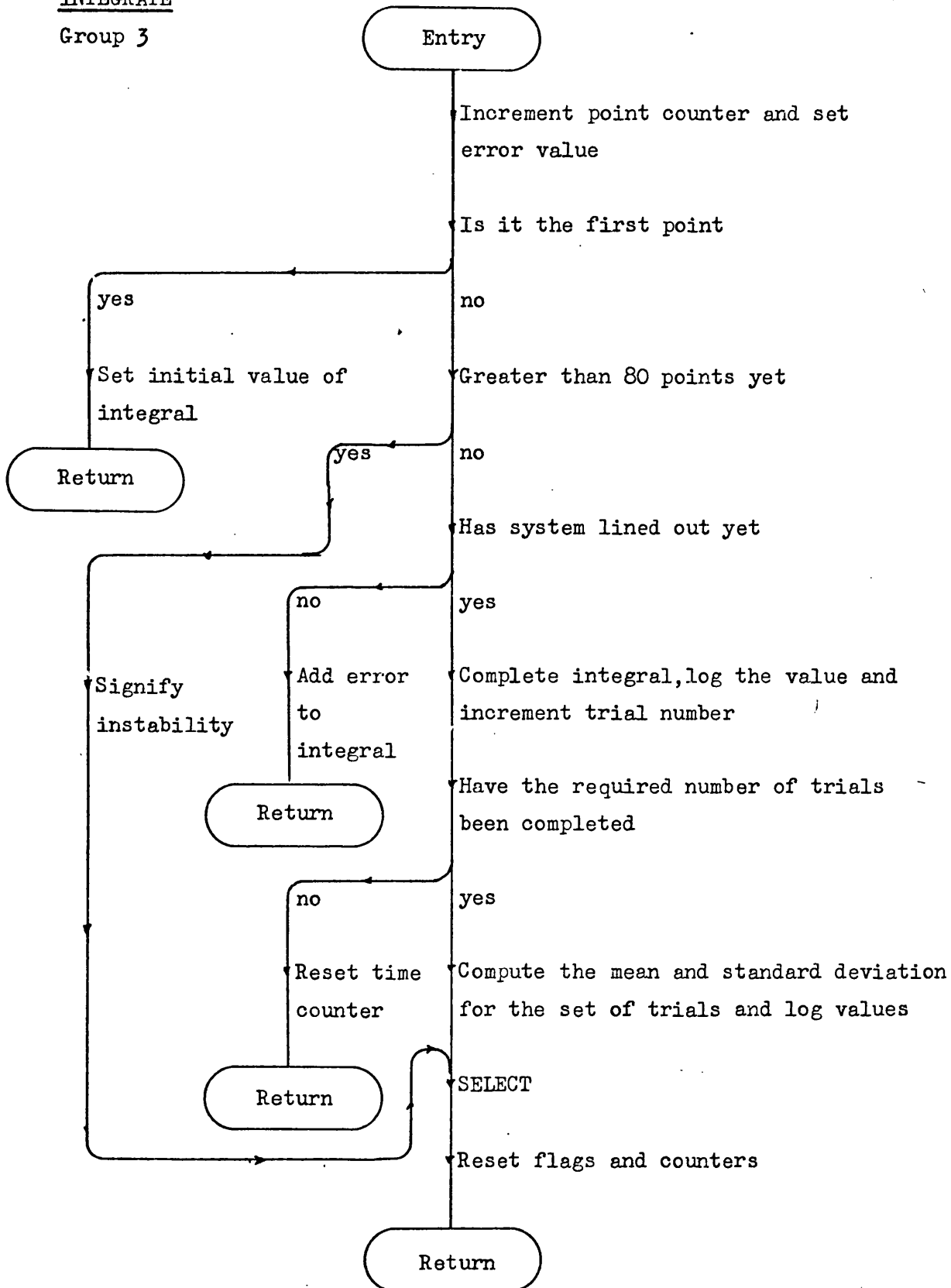


Figure A3. Flowsheet of Lattice Search Program

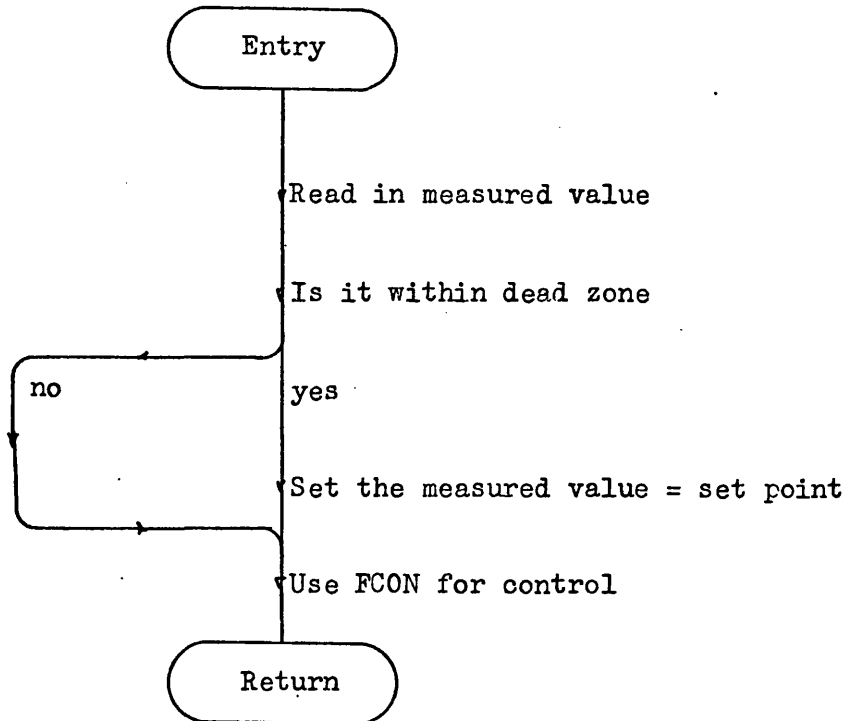
INTEGRATE

Group 3

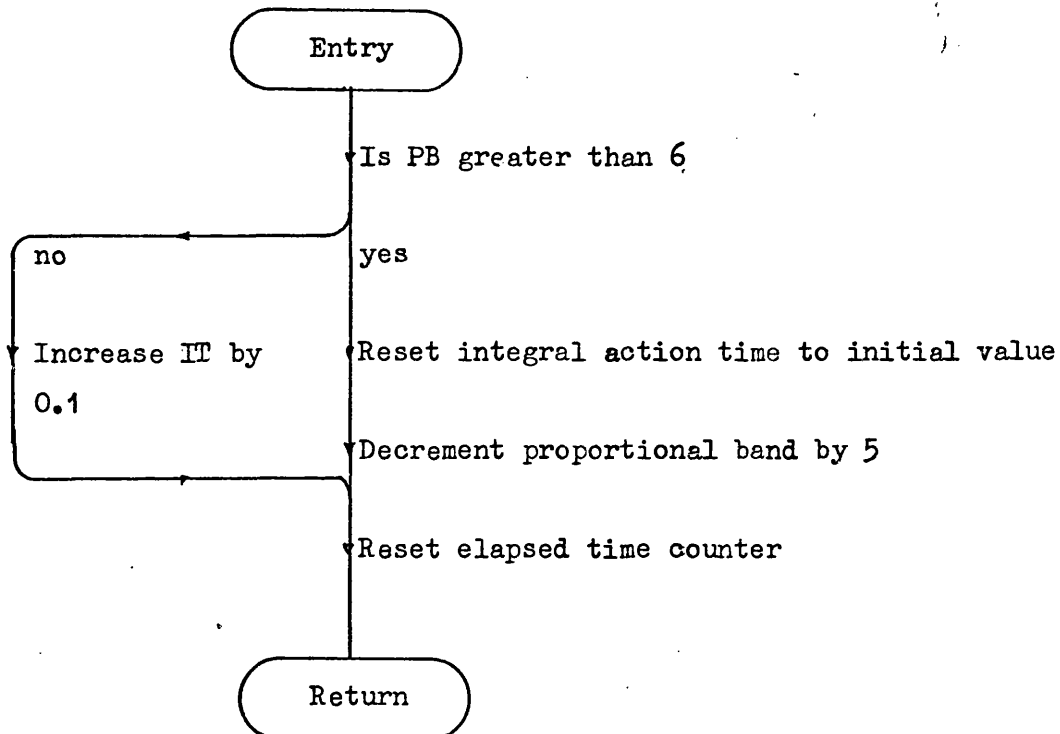
Figure A3. Continued

CONTROL

Group 6

SELECT

Group 12

Figure A3. Continued

<u>Program Section</u>	<u>Function</u>
Lines 1.01 to 1.09	Conversational mode of input for program initialization, also provides a hard copy log of system parameters
Lines 1.11 to 1.16	Clearing of counters and flags
Lines 1.20 to 1.23	Timing loop for program synchronization
Line 1.25	D0 subroutine call for I/O functions see group 6
Lines 1.27 to 1.62	Performs the logical operations of the program, deciding upon the next operation required
Group 2	Computation of standard deviations
Lines 3.01 to 3.34	Trapezoidal integration, line out test and instability exit
Lines 3.40 to 3.48	Completion of error integration, logging results and reset of flags and counters for next attempt. Also tests for completion of set number of experiments
Lines 3.50 to 3.64	Computation of mean and standard deviation for upward step changes and downward step changes. Logging results for the set of experiments
Line 3.66	D0 subroutine call to select a new lattice point

Figure A4. Description of Lattice Search Program

<u>Program Section</u>	<u>Function</u>
Lines 3.68 to 3.70	Reset counters and return for logical operations
Group 6	Data input and output using new FOCAL functions
Group 8	Pre-program initialization of user scan flags
Group 12	Selection of next lattice point for the search

Figure A4. continued

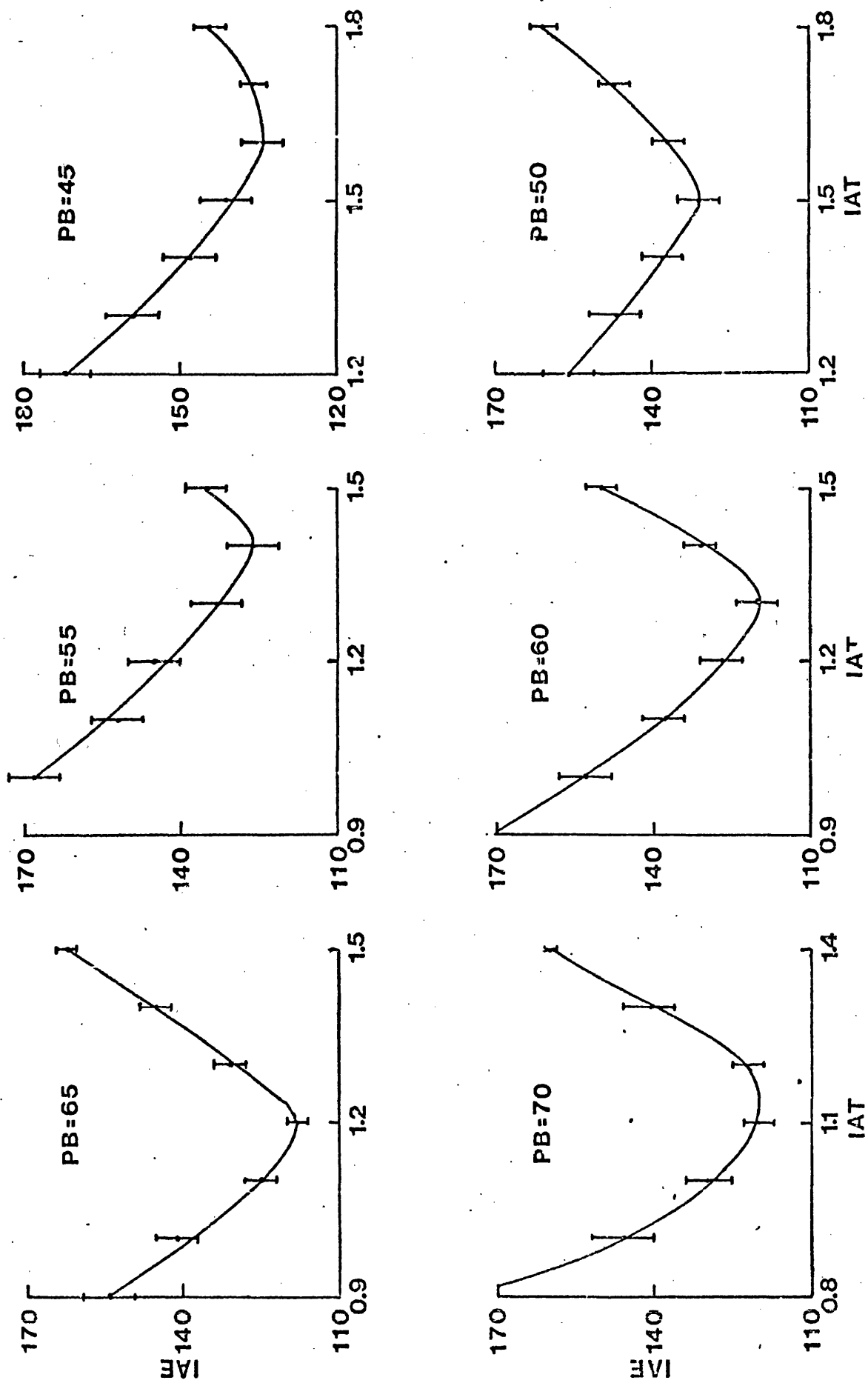


Figure A5.

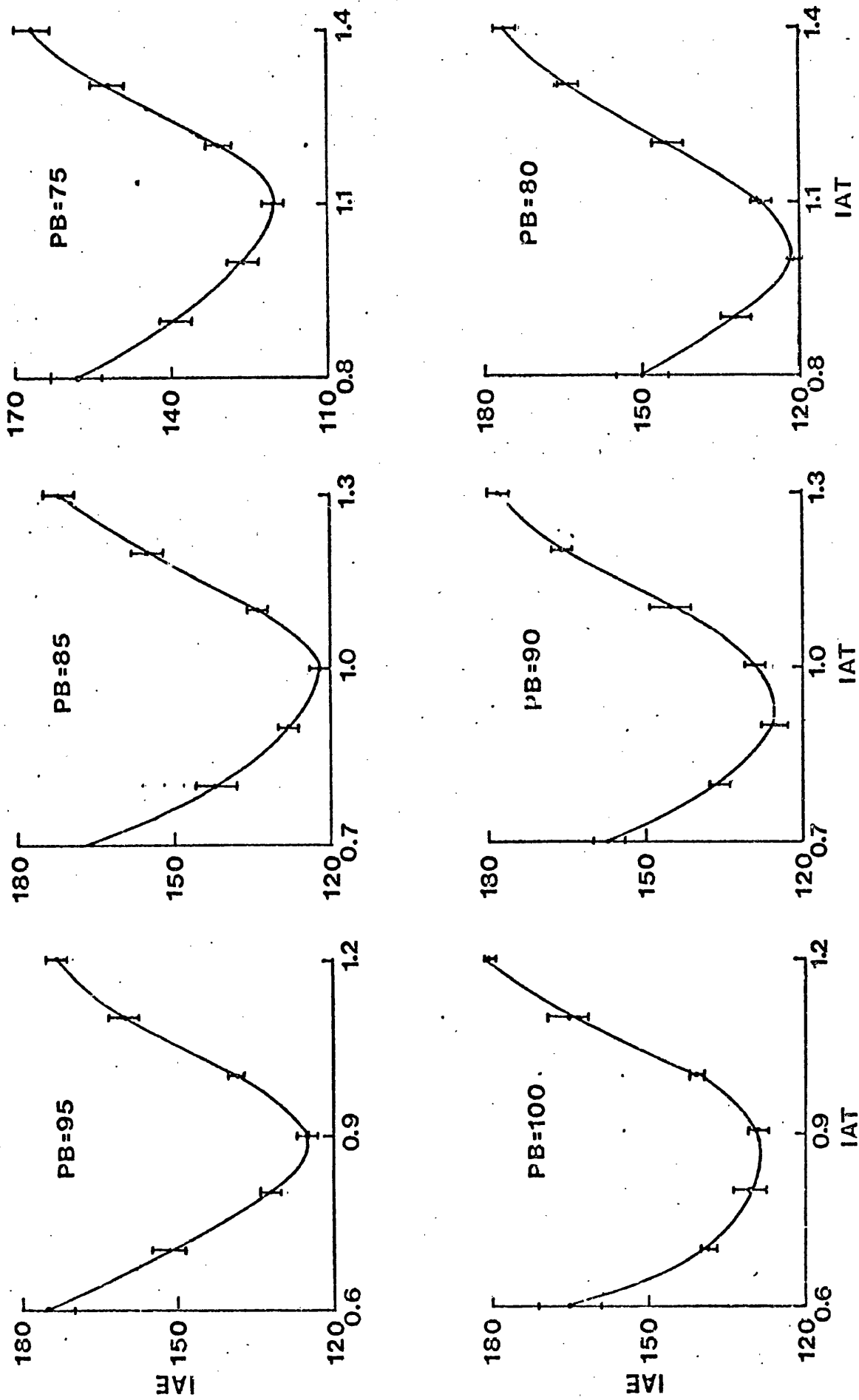


Figure A6.

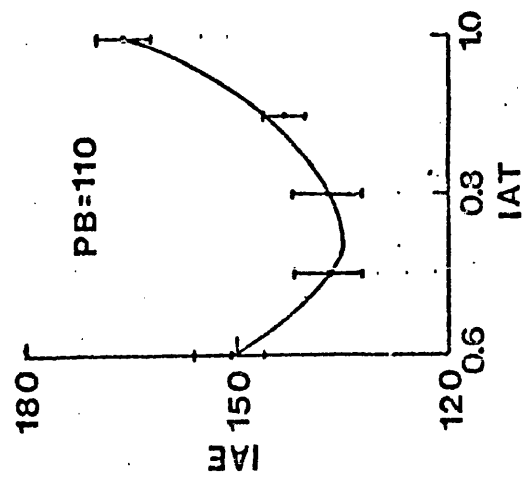
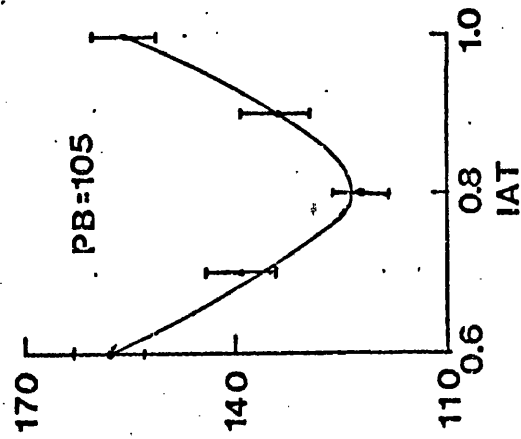
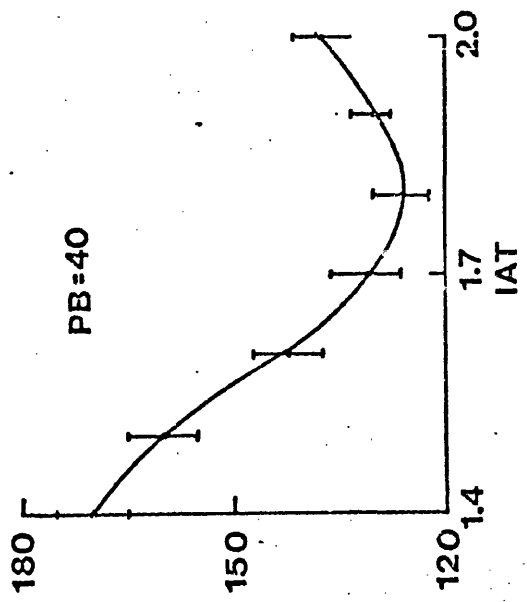
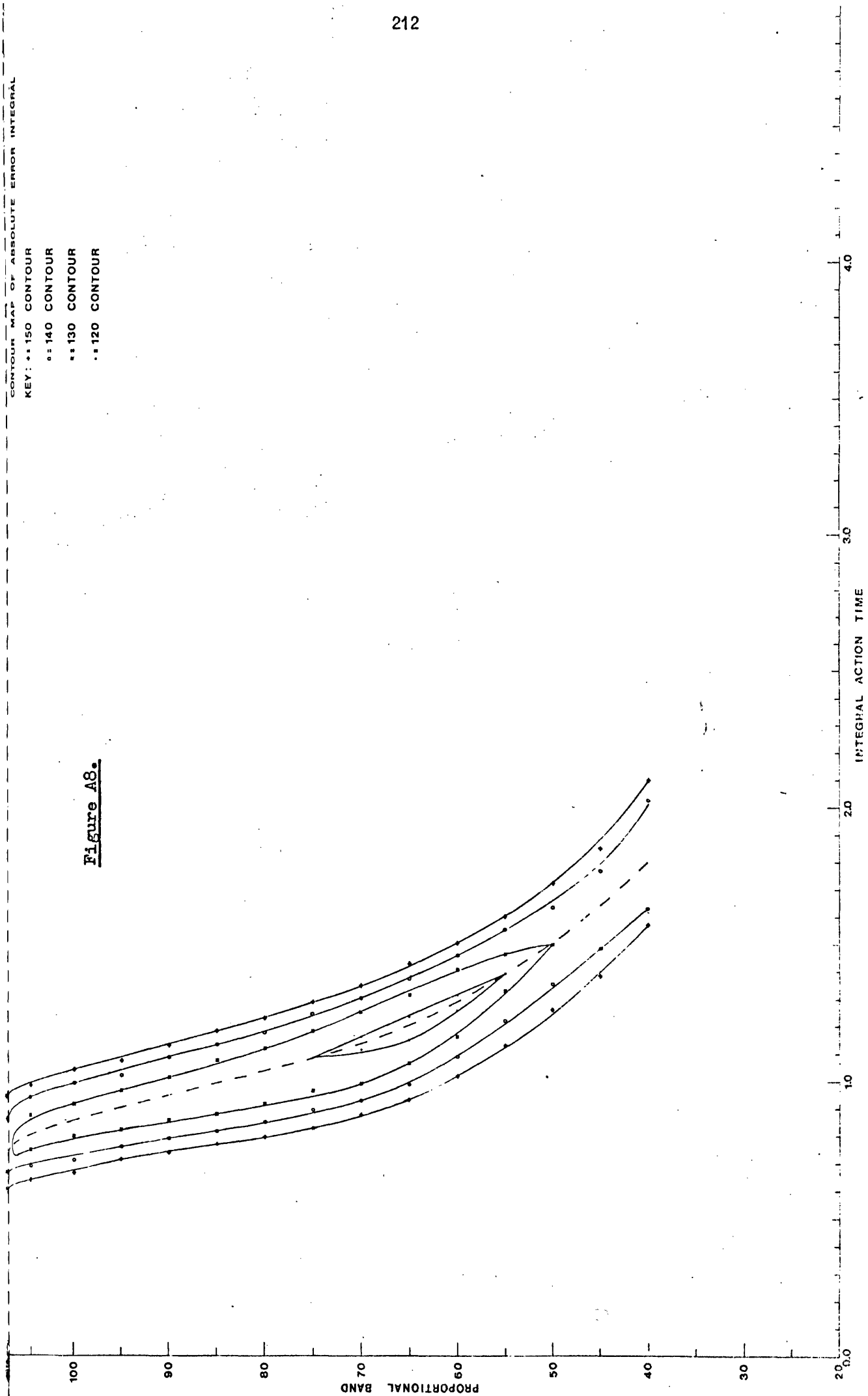


Figure A7.

Figure A8.

C-8K SYMBOLS FOCAL ©1974 NEW TTY

```

01.01 A "INPUT CODE"IC,"INPUT CHANNEL ADDRESS"CI,!
01.03 A "LOOP NUMBER"L,"OUTPUT CHANNEL ADDRESS"CO,!
01.05 A "SCAN INTERVAL"SC,"DEAD ZONE"DZ,"LINE OUT TRIALS"LO,!
01.06 A "DEAD ZONE FOR INTEGRAL"DZ(1),!
01.07 A "INITIAL SET POINT"IS,"FINAL SET POINT"HS,!
01.09 T "INITIAL CONDITIONS",!
01.10 S XL(1)=0;S XL(3)=0;S XL=0;S PB(2)=50;S IT(2)=2
01.11 S PF(3)=50;S IT(3)=2
01.12 D 3.12
01.13 D 2.0
01.15 S PB=PB(1);S IT=IT(1);S KR=1;S GF=-1;S SP=IS;S KC=100/PB
01.20 I (FLAG(SF(1)))1.25
01.22 G 1.20
01.25 D 6.0
01.28 I (GF)1.30,1.30,1.50
01.30 D 3.0
01.32 G 1.20
01.50 D 10.0
01.52 G 1.20

```

Figure A9. FOCAL Program for Simplex Search Procedure

```

02.01 A "STARTING POINT",I,"PROP BAND"PB(1),"IAT"IT(1),I
02.02 A "INTEGRAL ACTION STEP SIZE"DI,"APPROX CHANGE IN PROP BAND"DP,I
02.03 D 1.15
02.05 S GS=-1;S GF=0
02.07 I (FLAG(SF(1)))2.10
02.03 G 2.07
02.10 D 6.0
02.12 I (GF)2.40,2.20,2.30
02.20 D 3.0
02.22 G 2.07
02.30 D 10.0
02.32 G 2.07
02.40 I (GS)2.52,2.60
02.41 S KR=2
02.42 I (FAPS(SM(2))-SM(20))-5)2.70,2.70
02.44 S DP=DP*(SM(20)-SM(1))/(SM(2)-SM(1))
02.46 G 2.64
02.52 S IT(2)=IT(1)-DI;S PB(2)=PB(1)
02.54 S PB=PB(2);S IT=IT(2);S GS=0;S GF=0;S KG=100/PB;S KR=2
02.56 G 2.07
02.60 S KR=2;S SM(20)=SM(2);S IT(3)=IT(1)
02.64 S PB(3)=PB(1)+DP;S IT=IT(3);S PB=PB(3);S GF=0;S GS=1;S KG=100/PB
02.66 G 2.07
02.70 S PB(2)=((PB(3)-PB(1))*0.2587/0.9657)+PB(1)
02.72 S IT(3)=((IT(2)-IT(1))*0.2587/0.9657)+IT(1)
02.74 S GS=8;R

```

Figure A9. continued

```

03.01 I (X-IS)3.03,3.10
03.03 S CC(10)=CC(10)+1;S CC(20)=0
03.05 I (CC(10)-100)3.07,3.20,3.20
03.07 R
03.10 S CC(20)=CC(20)+1;I (CC(20)-10)3.07
03.12 S CC(10)=0;S CC(20)=0
03.14 S GF=1;S SP=HS;S SM=0;S J=0
03.16 R
03.20 T "UNSTABLE AT LOWER SET POINT",!,"PROP BAND"PB,"IAT"IT,!
03.22 S CC(20)=0;S CC(10)=0
03.24 I (GS-8)3.30,3.40
03.30 T "PLEASE START AGAIN AT A SUITABLE SET POINT",!
03.32 Q
03.40 S SM(A)=SM(B)+SM(C);D 4.0
03.42 S CC(30)=CC(30)+1;I (A-XL)3.46,3.44,3.46
03.44 S XL=B;S B=A;S A=XL
03.46 D 11.0;R

04.01 I (SM(1)-SM(3))4.20,4.20
04.02 I (SM(3)-SM(2))4.10,4.10
04.04 S A=1;S B=3;S C=2
04.06 R
04.10 I (SM(2)-SM(1))4.15,4.15
04.12 S A=2;S B=1;S C=3
04.14 R
04.15 S A=1;S B=2;S C=3
04.16 R
04.20 I (SM(3)-SM(2))4.30,4.30
04.22 I (SM(2)-SM(1))4.26,4.26
04.24 S A=3;S B=2;S C=1
04.25 R
04.26 S A=3;S B=1;S C=2
04.28 R
04.30 S A=2;S B=3;S C=1
04.32 R

06.01 S X=FIV(IC,CI)
06.03 I (FABS(X-SP)-DZ)6.05,6.05,6.07
06.05 S X=SP
06.07 S Z=FCON(L,X,SP,IT,SC,KC,CO)
06.08 R

```

Figure A9. continued

```

08.01 S SF(1)=10;S SF(2)=20;S SF(3)=30;S N=10
08.03 S Z=FLAG(0ST,N,SF(1),SF(2),SF(3))
08.05 Q

10.01 S J=J+1;S E=FABS(X-SP)
10.03 I (1-J)10.10
10.05 S SM=E
10.07 R
10.10 I (80-J)10.30
10.11 I (DZ(1)-E) 10.20
10.12 S E=0
10.14 S CT=CT+1
10.16 I (LO-CT)10.50
10.18 G 10.22
10.20 S CT=0
10.22 S SM=SM+2*E
10.24 R
10.30 T "UNSTABLE",1,"PROP BAND"PB,"IAT",IT,1
10.32 I (GS-8)10.34,10.36
10.34 T "PLEASE START AGAIN",!!!!;G
10.36 S SM(A)=SM(B)+SM(C);G 10.62
10.50 S SM=(SM+E)*SC/2
10.52 T "PROP BAND"PB,"IAT"IT,"GAIN"KG,"ERROR INTEGRAL"SM,1
10.54 I (2-KR)10.60
10.56 S SM(KR)=SM;S KR=KR+1;S PB=PB(KR);S IT=IT(KR);S A=3
10.58 S SP=IS;S KG=100/PB;S GF=-1;S SM=0;S J=0
10.59 R

```

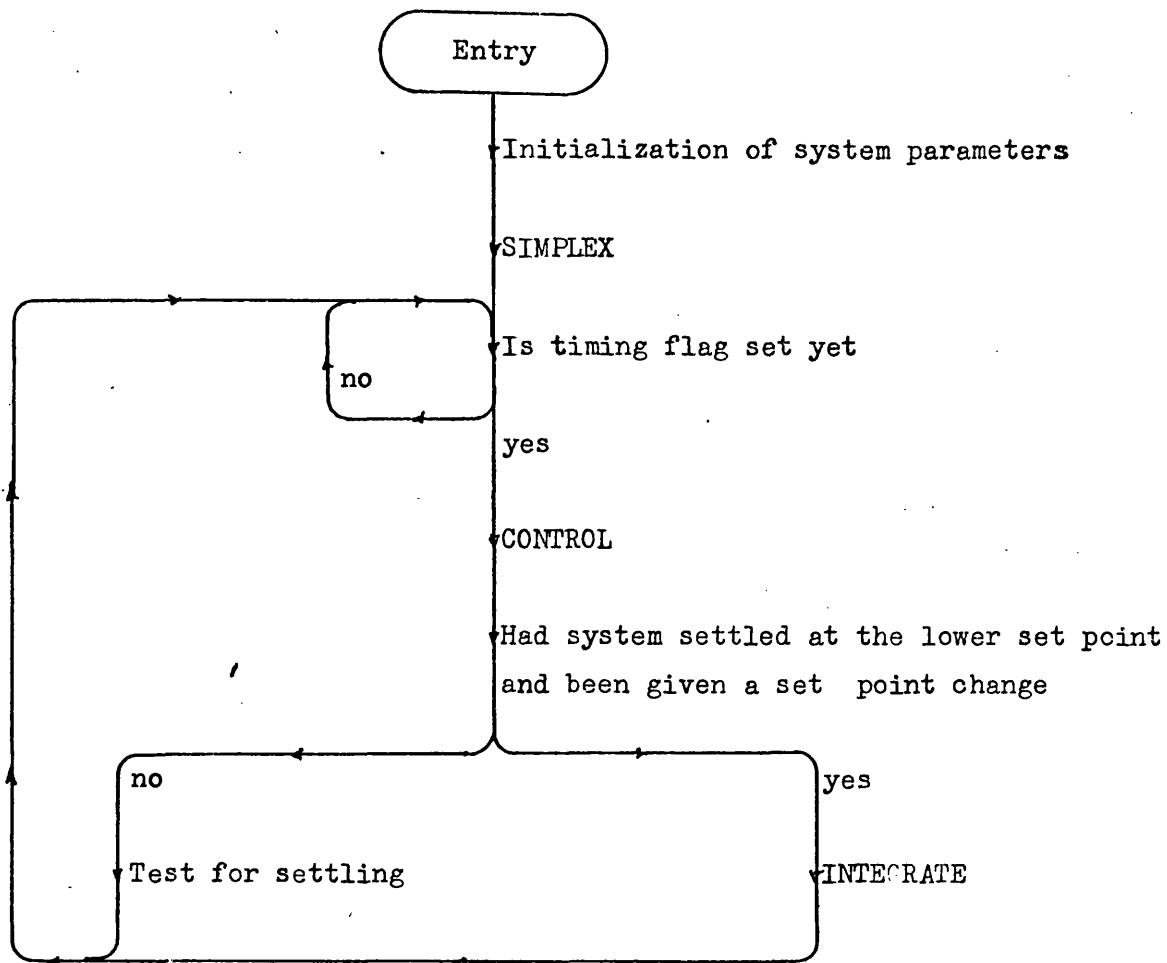
Figure A9. continued

```

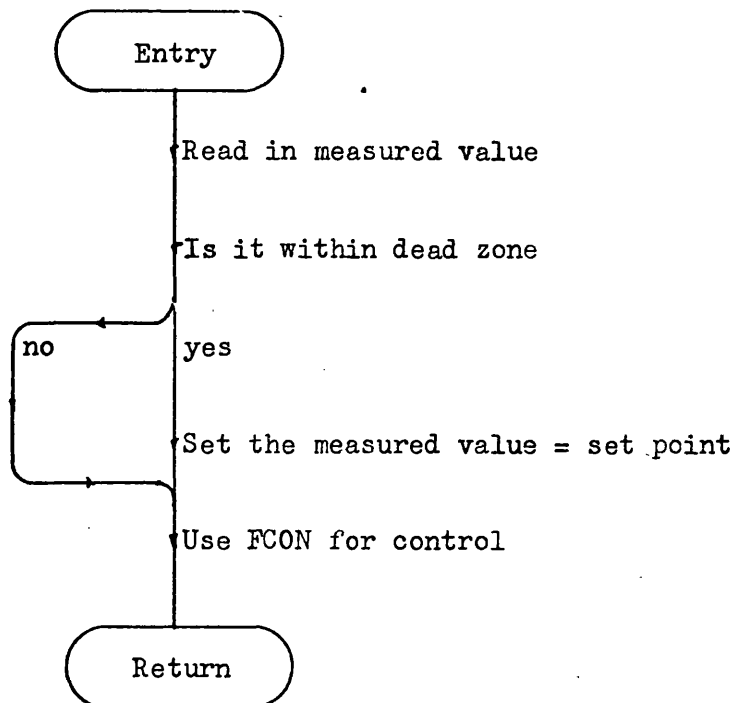
10.60 S SM(A)=SM
10.62 D 4.0
10.64 I (C-XL(3))10.68,10.80,10.68
10.68 S XL(3)=C;S CC(30)=1
10.70 I (A-XL)10.92,10.90,10.92
10.80 S CC(30)=CC(30)+1;I (CC(30)-12)10.70
10.82 S PB(A)=(PB(A)+PB(C))/2;S IT(A)=(IT(A)+IT(C))/2
10.84 S PB(P)=(PB(P)+PB(C))/2;S IT(P)=(IT(P)+IT(C))/2
10.86 T "STEP LENGTH CHANGE",I;D 1.15
10.88 S XL(3)=0;S XL(1)=0;S XL=0
10.89 R
10.90 S XL=B;S B=A;S A=XL
10.92 D 11.0;R

11.01 S PB(4)=PB(A);S IT(4)=IT(A);S SM(4)=SM(A);S XL(1)=XL;S XL=A
11.03 S PB(A)=PB(P)+PB(C)-PB(A);S IT(A)=IT(P)+IT(C)-IT(A)
11.05 S PB=PB(A);S IT=IT(A)
11.07 S SP=IS;S KG=100/PB;S GF=-1;S SM=0;S J=0
11.09 R
*
```

Figure A9. continued

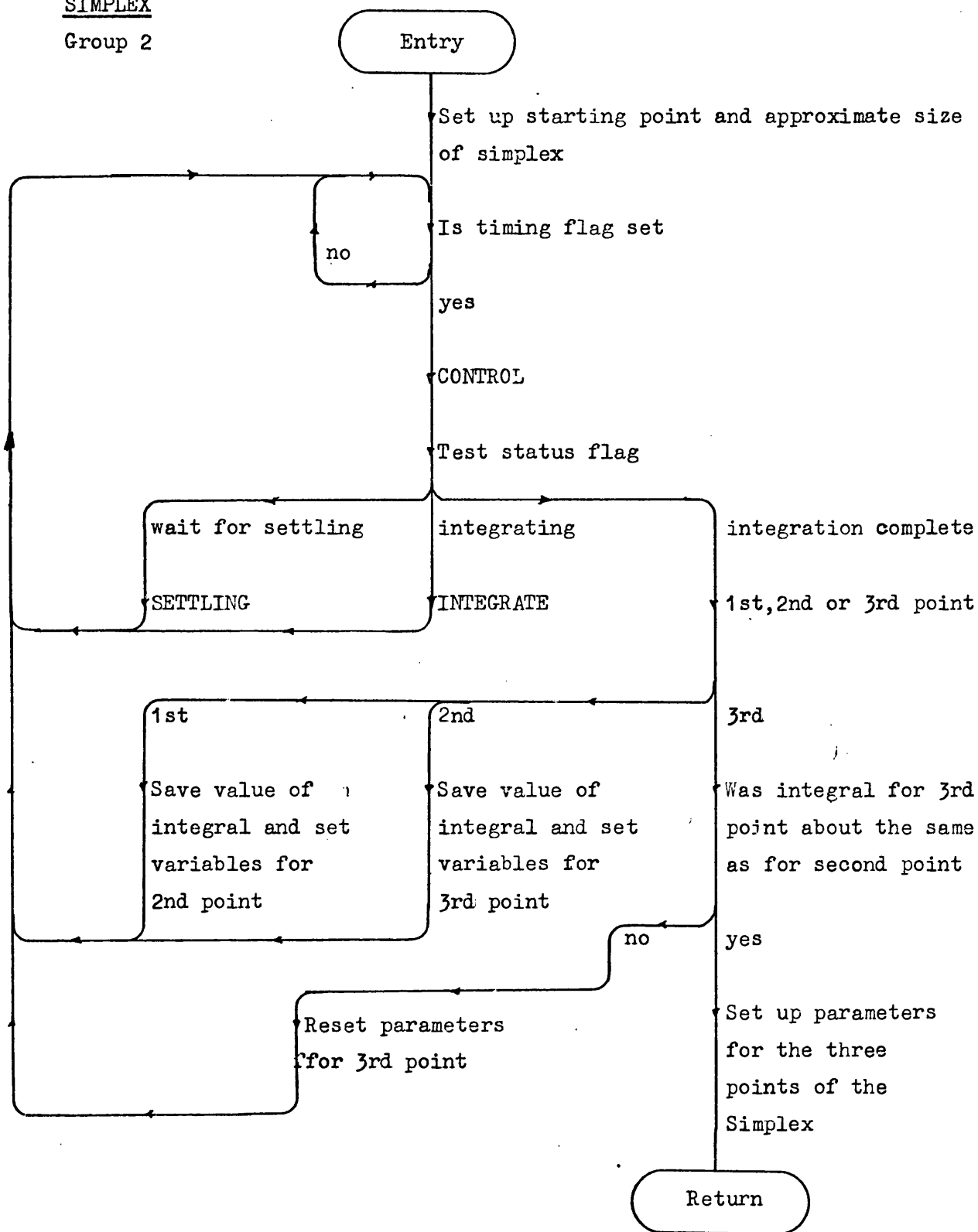
CONTROL

Group 6

Figure A10. Flowsheet of Simplex Search Program

SIMPLEX

Group 2

Figure A10. Continued

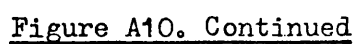
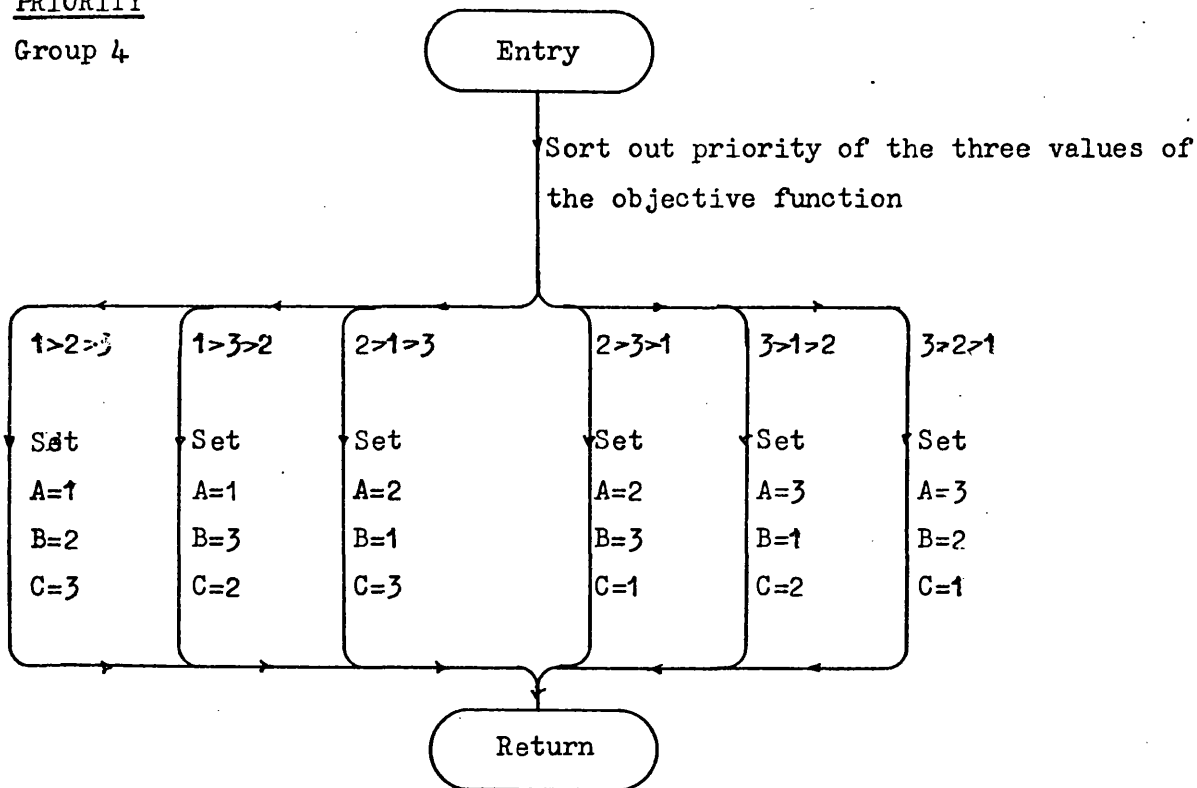


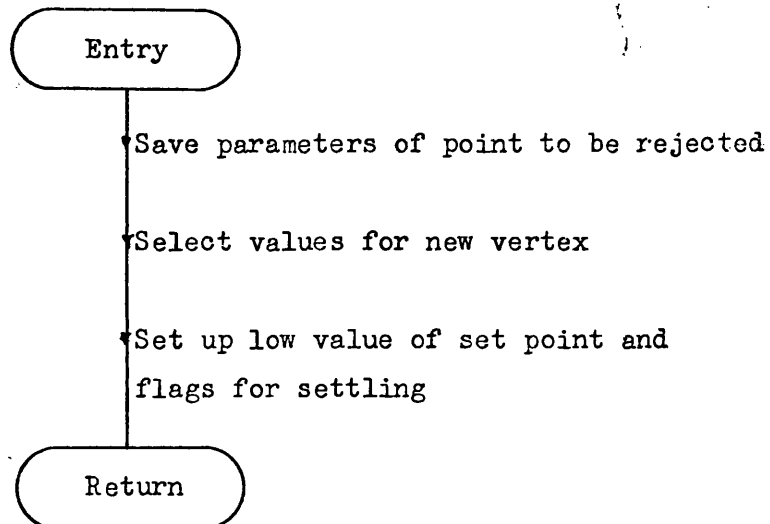
Figure A10. Continued

PRIORITY

Group 4

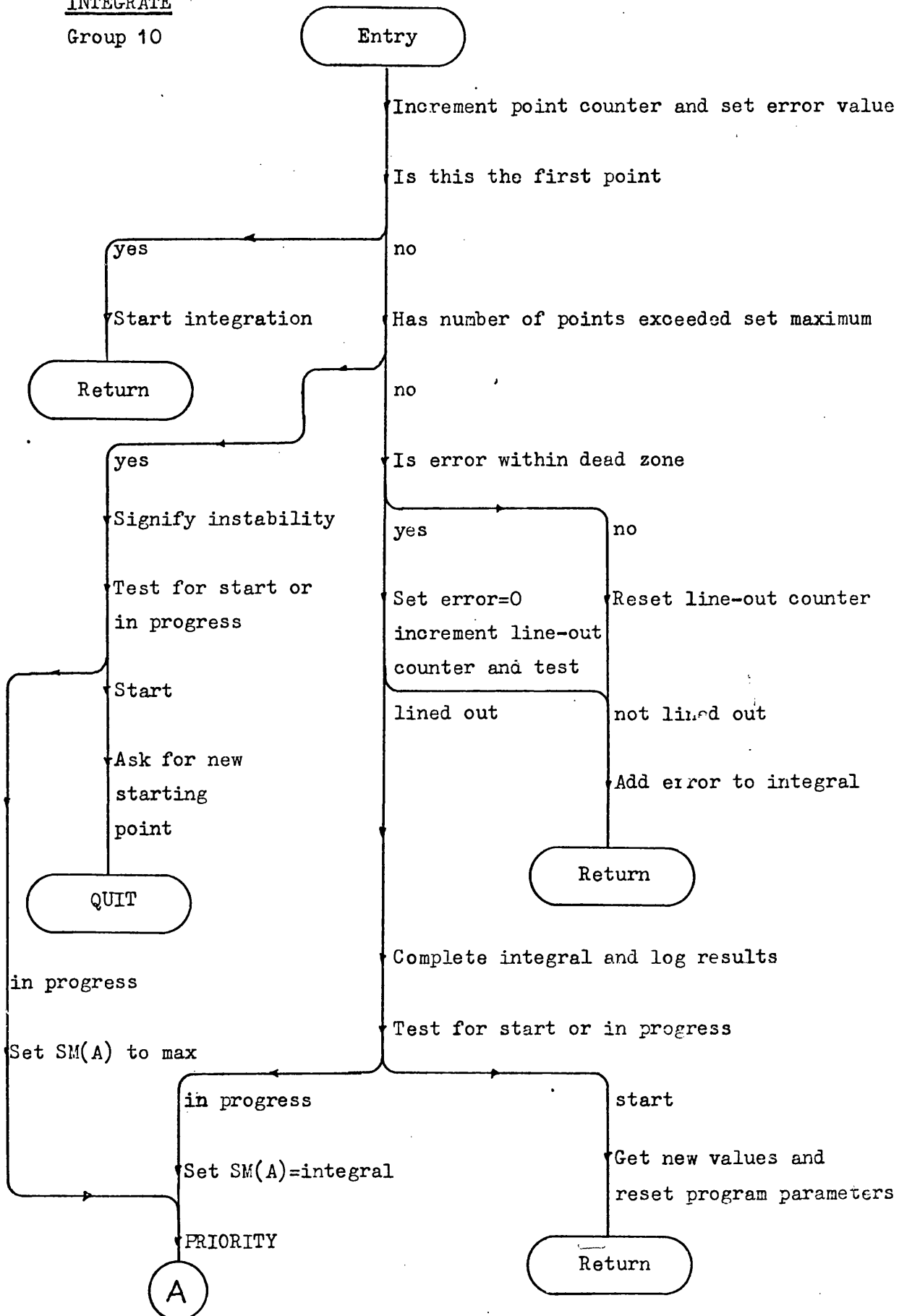
SELECT

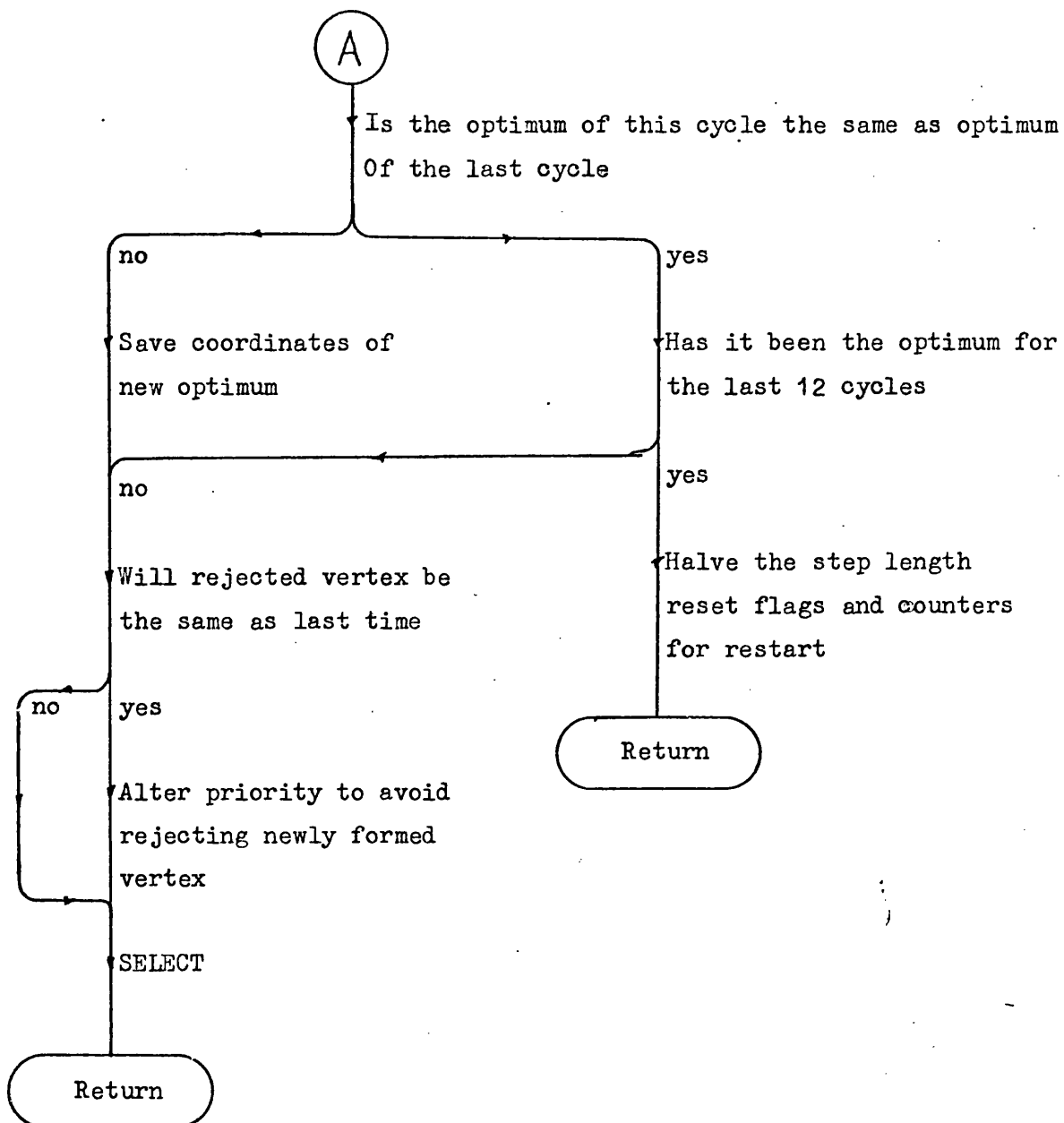
Group 11

Figure A10. Continued

INTEGRATE

Group 10

Figure A10. Continued

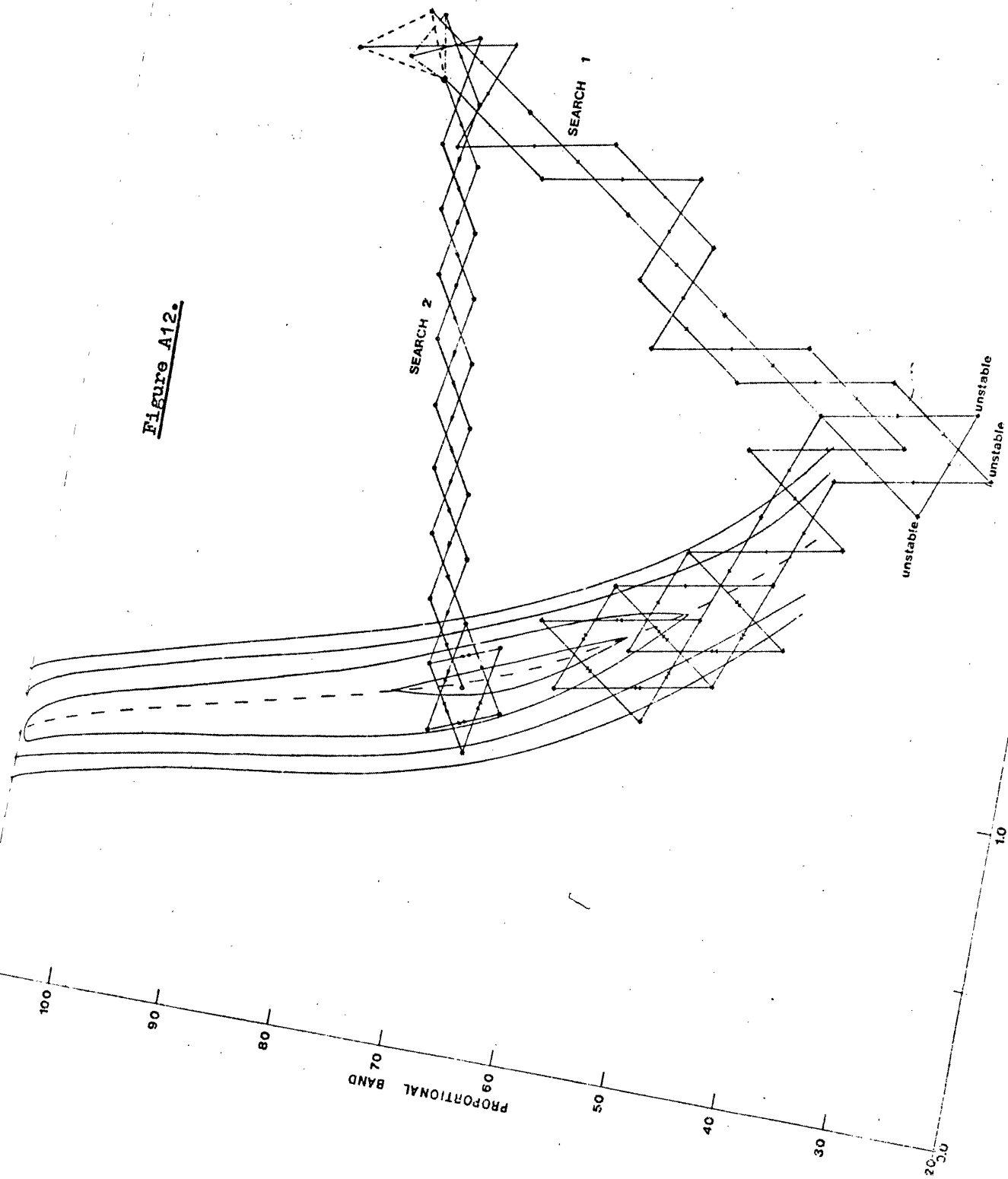
Figure A10. Continued

<u>Program Section</u>	<u>Function</u>
Lines 1.01 to 1.09	Conversational mode of input for program initialization and to provide a hard copy log of system parameters
Lines 1.10 to 1.12	Reset of flags and counters
Line 1.13	D0 subroutine call to group 2 for setting up the initial Simplex
Lines 1.15 to 1.52	Timing and sequence control of system
Group 2	Simplex initialization by examination of local gradients
Group 3	Lower set point line out testing and instability exit
Group 4	Grading of Objective Function values for vertex rejection procedure
Group 6	Data input and control
Group 8	Pre-program initialization of user scan flags
Group 10	Error integration routine
Group 11	New vertex selection procedure

Figure A11. Description of Simplex Search Program

Figure A12.

INITIAL STAGES OF TWO SIMPLEX SEARCHES



```

C-8K SYMBOLS FOCAL 01974 NEW TTY

01.01 A "INPUT CODE"IC,"INPUT CHANNEL ADDRESS"CI,!
01.03 A "LOOP NUMBER"L,"OUTPUT CHANNEL ADDRESS"CO,!
01.05 A "SCAN INTERVAL"SC,"DEAD ZONE"DZ,"LINE OUT TRIALS"LO,!
01.07 A "INITIAL SET POINT"IS,"FINAL SET POINT"HS,!
01.09 T "INITIAL CONDITIONS",!
01.10 A "INTEGRAL ACTION TIME"IT,"PROPORTIONAL BAND"PB,!
01.11 A "MAX STEP FOR IAT"DI,"MAX STEP FOR PROP BAND"DP,!
01.12 T "RANDOM SEARCH LOADED",!
01.13 S PR(20)=0
01.15 S KR=1;S CC(10)=0;S CC(20)=0
01.17 S KG=100/PR;S SP=IS;S GF=-1
01.20 I (FLAG(SF(1)))1.25
01.22 G 1.20
01.25 D 6.0
01.28 I (CF)1.30,1.35,1.50
01.30 I (X-IS)1.32,1.35,1.32
01.32 S CC(10)=CC(10)+1;S CC(20)=0
01.33 I (CC(10)-100)1.20,1.80,1.80
01.35 S CC(20)=CC(20)+1
01.36 I (CC(20)-10)1.20
01.37 S GF=0
01.40 S SP=HS;S CC(10)=0;S CC(20)=0;S GF=1;S SM=0;S J=0
01.42 G 1.20
01.50 D 9.0
01.52 G 1.20
01.80 T "UNSTABLE AT LOWER SET POINT",!,"PROP BAND"PB,"IAT"IT,!
01.82 S CC(10)=0;S CC(20)=0
01.84 I (1-KR)1.89
01.86 S W=0
01.87 D 11.0
01.88 G 1.20
01.89 S W=10
01.90 G 1.87

```

Figure A13. FOCAL Program for Random Search Procedure

```

06.01 S X=FIN(IC,CI)
06.03 I (FABS(X-SP)-1)6.05,6.05,6.07
06.05 S X=SP
06.07 S Z=FCON(L,X,SP,IT,SC,KG,CO)
06.09 R

08.01 S SF(1)=10;S SF(2)=20;S SF(3)=30;S N=10
08.03 S Z=FLAG(OST,N,SF(1),SF(2),SF(3))
08.05 Q
08.20 S CT=0

09.01 S J=J+1;S E=FABS(X-SP)
09.03 I (1-J)9.10
09.05 S SM=E
09.07 R
09.10 I (80-J)9.30
09.12 I (DZ-E)9.20
09.13 S E=0
09.14 S CT=CT+1
09.16 I (LO-CT)9.50
09.18 G 9.22
09.22 S SM=SM+2*E
09.24 R

```

Figure A13. continued

```

09.30 T "UNSTABLE",!,Z5.3,"PROP BAND"PB,"IAT"IT,!
09.32 I (1-KR)9.61
09.34 S WW=0
09.36 D 11.0
09.38 G 9.66
09.50 S SM=(SM+E)*SC/2
09.51 T "P E"PB,"IAT,"IT,"ERROR "SM,!
09.52 I (1-KR)9.60
09.54 S KR=8
09.56 G 9.70
09.60 I (SM-SM(10))9.70
09.61 S MN=MN+1; I (15-MN)9.85
09.64 S WW=10
09.65 D 11.0
09.66 S SP=IS; S KC=100/PB; S GF=-1; S SM=0; S J=0
09.68 R
09.70 T Z5.3,"PROP BAND"PB,"IAT"IT,"ERROR INTEGRAL"SM,!
09.72 S SM(10)=SM; S PR(10)=PB; S IT(10)=IT; S MN=0
09.74 G 9.64
09.85 T "LAST ONE WAS MINIMUM WITHIN STEP SIZE",!
09.86 I (PR(20)-PR(10))9.92,9.88,9.92
09.88 S ZC=ZC+1; I (ZC-3)9.93
09.90 T "LAST ONE WAS BEST AFTER THREE EVALUATIONS",!
09.91 Q
09.92 S ZC=1; S PR(20)=PR(10)
09.93 T !!!
09.95 S MN=0; S KR=0; S IT=IT(10); S PB=PB(10)
09.97 G 9.66

11.01 S PR=PR(WW)+FRANC ) *DP
11.03 I (PB)11.01,11.01
11.05 S IT=IT(WW)+DI*FRANC )
11.07 I (IT)11.05,11.05
11.08 S KC=100/PB
11.09 R
*
```

Figure A13. continued

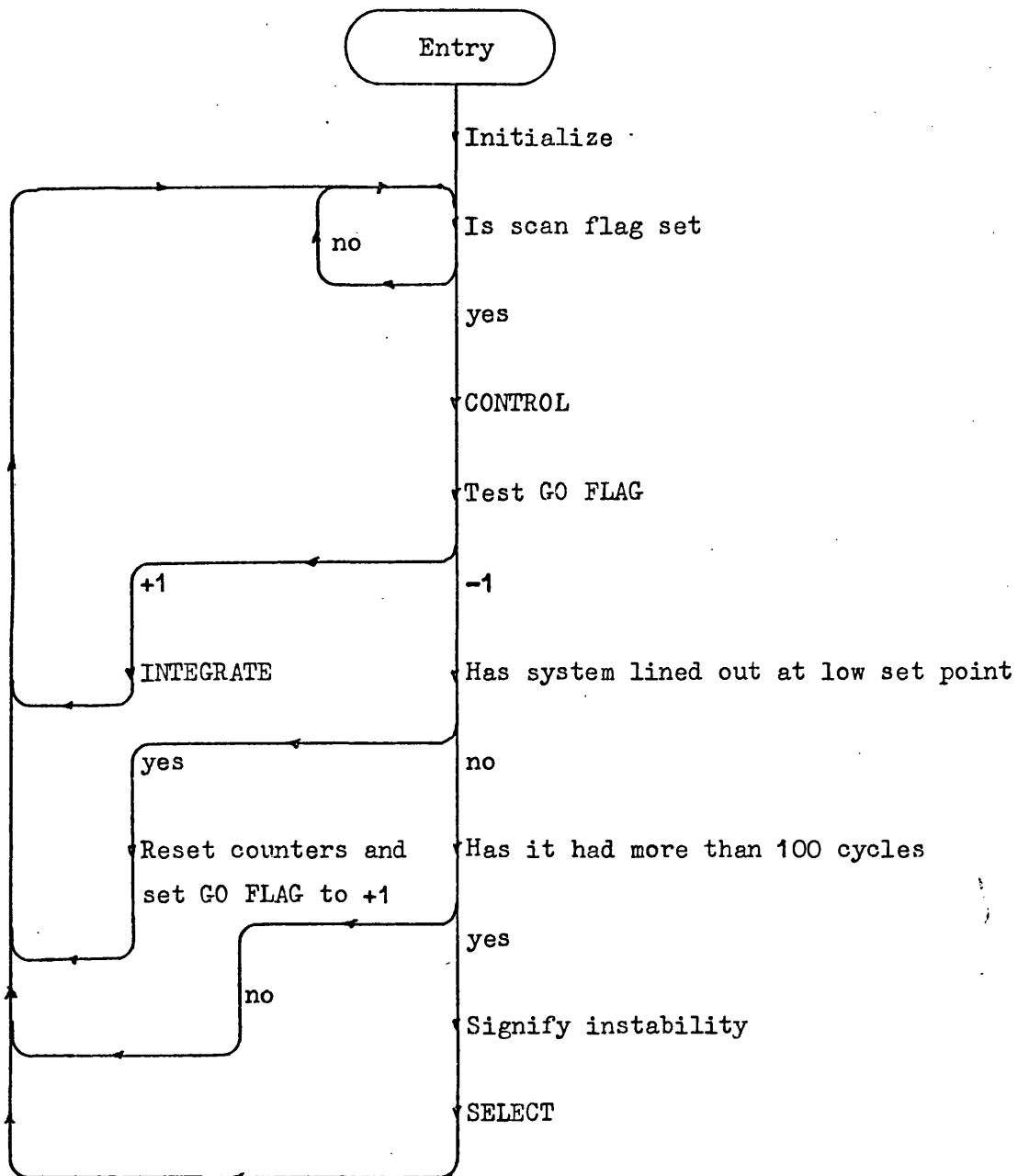
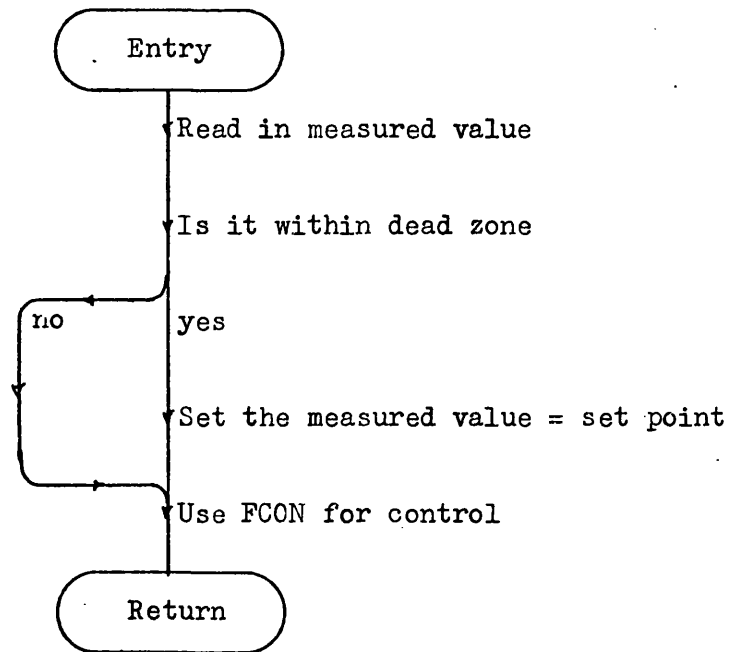


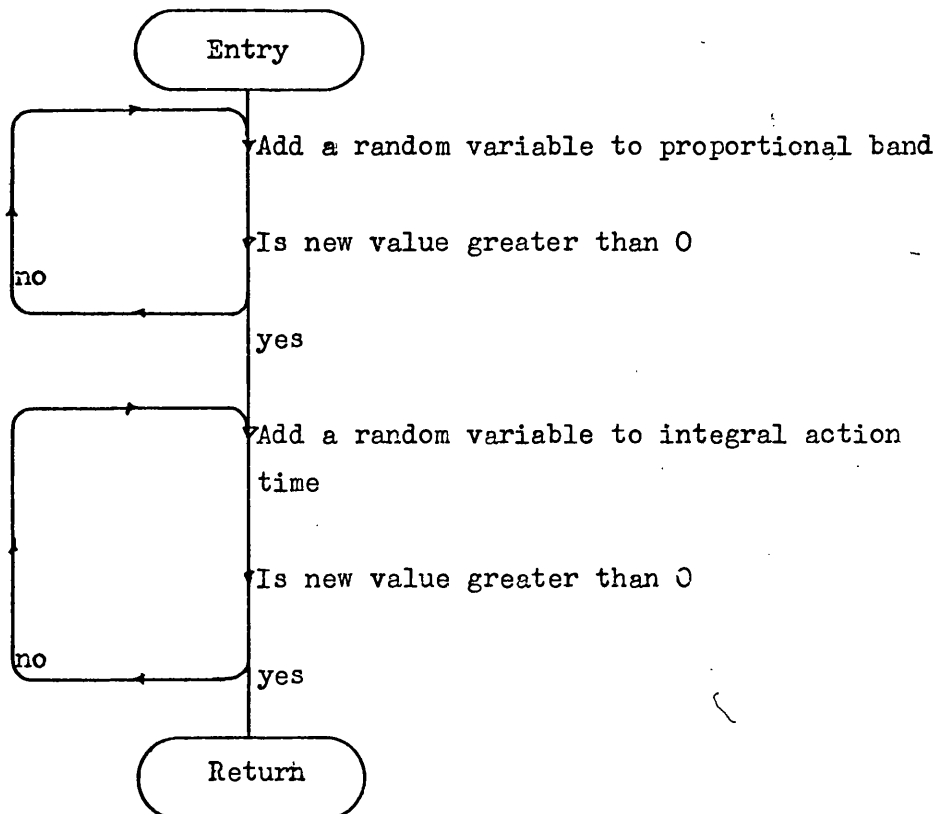
Figure A14. Flowsheet of Random Search Program

CONTROL

Group 6

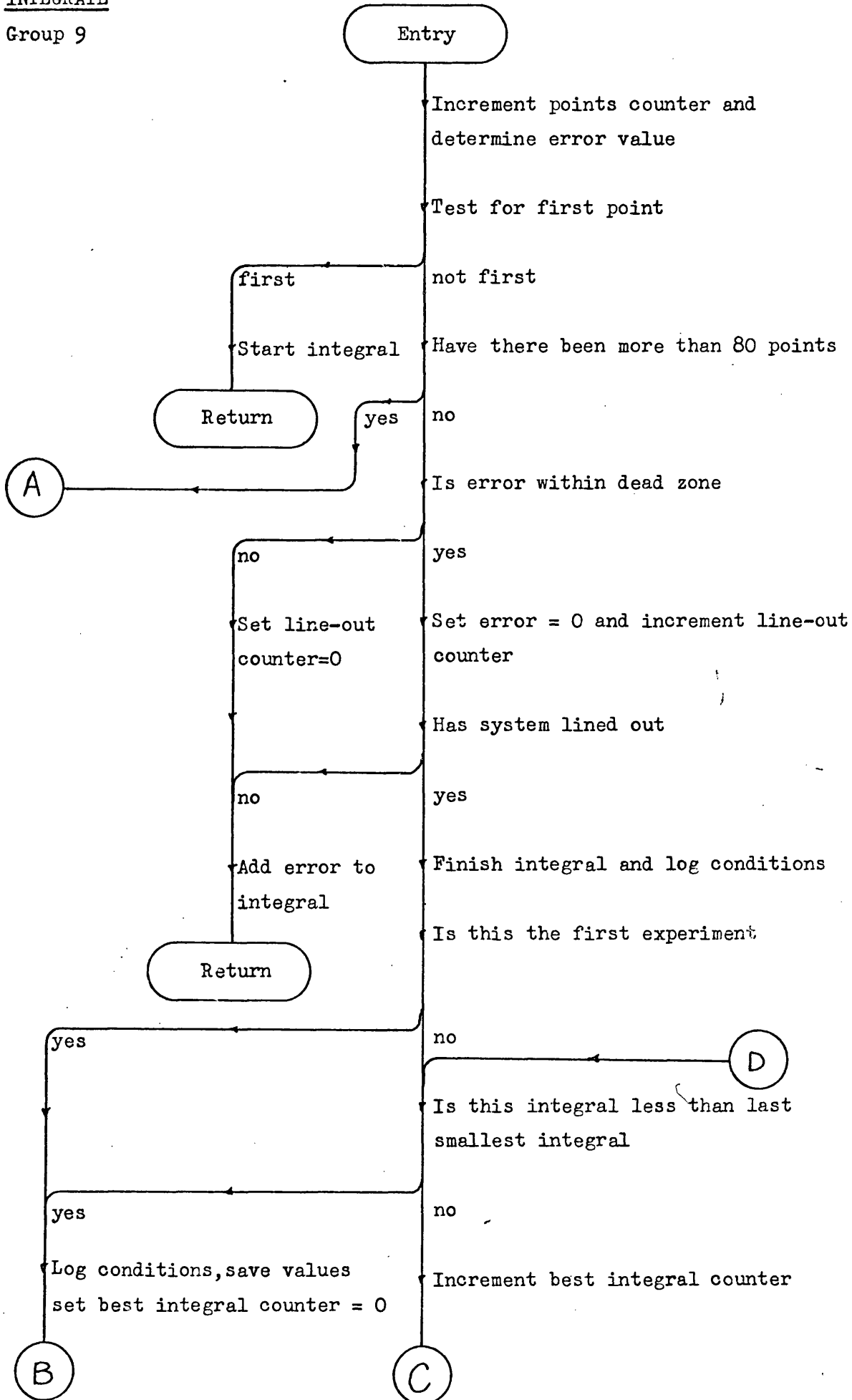
SELECT

Group 11

Figure A14. Continued

INTEGRATE

Group 9

Figure A14. Continued

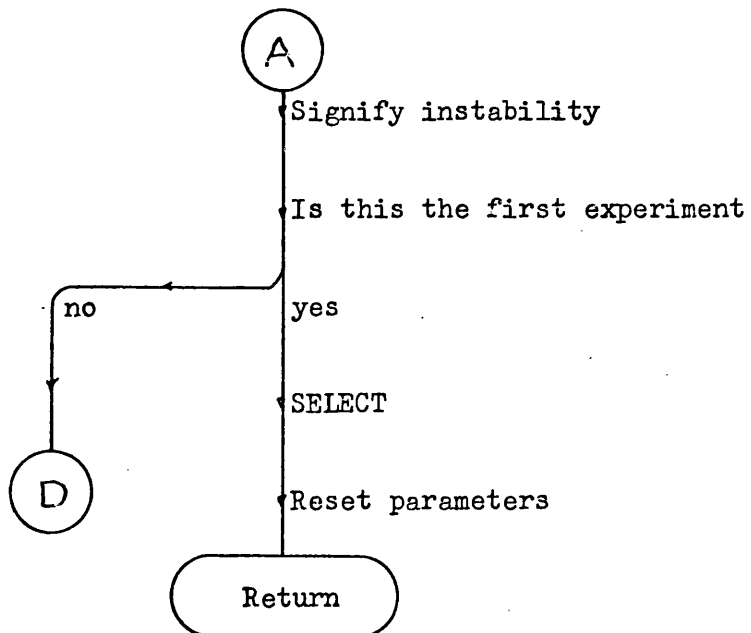
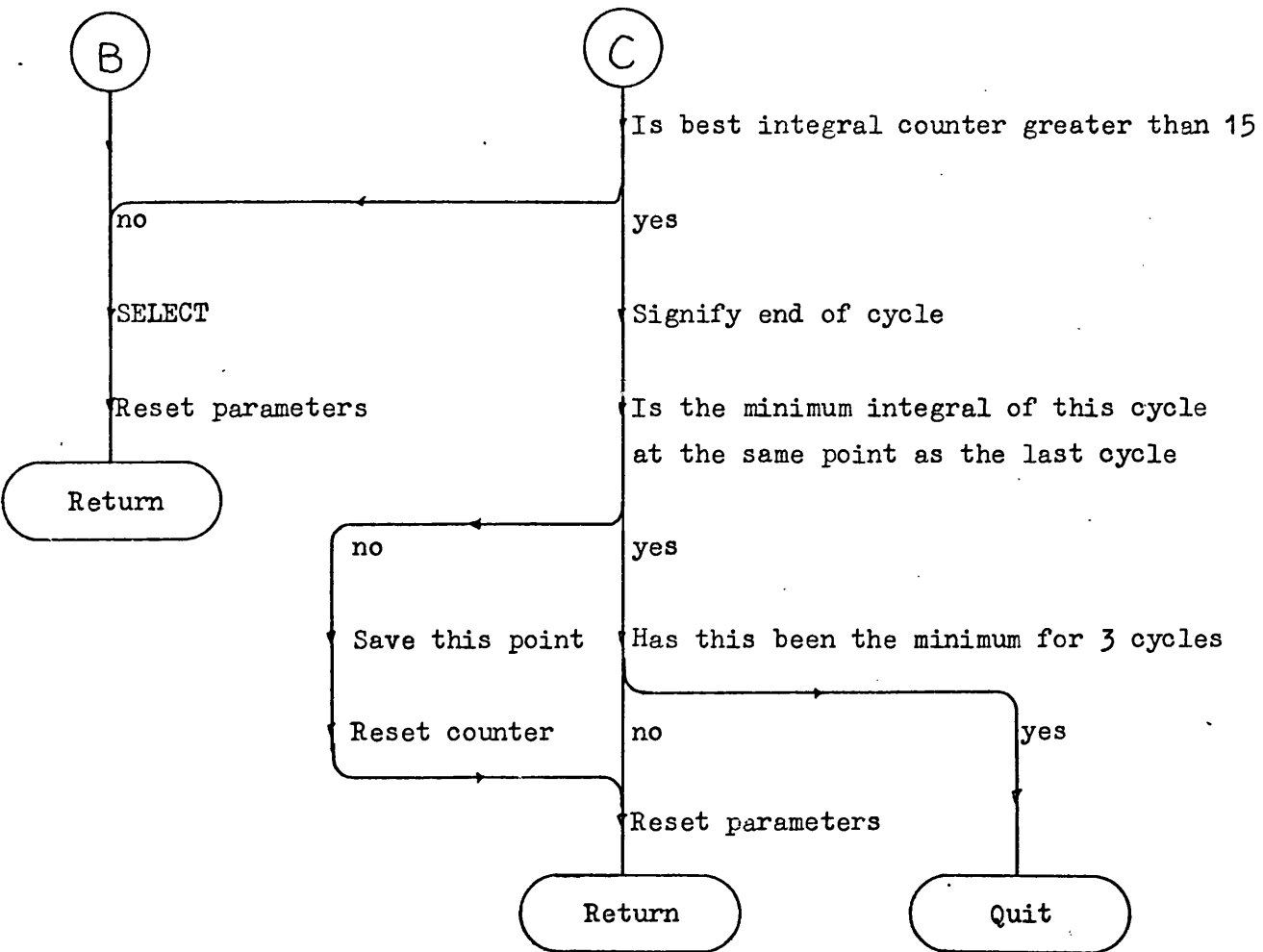


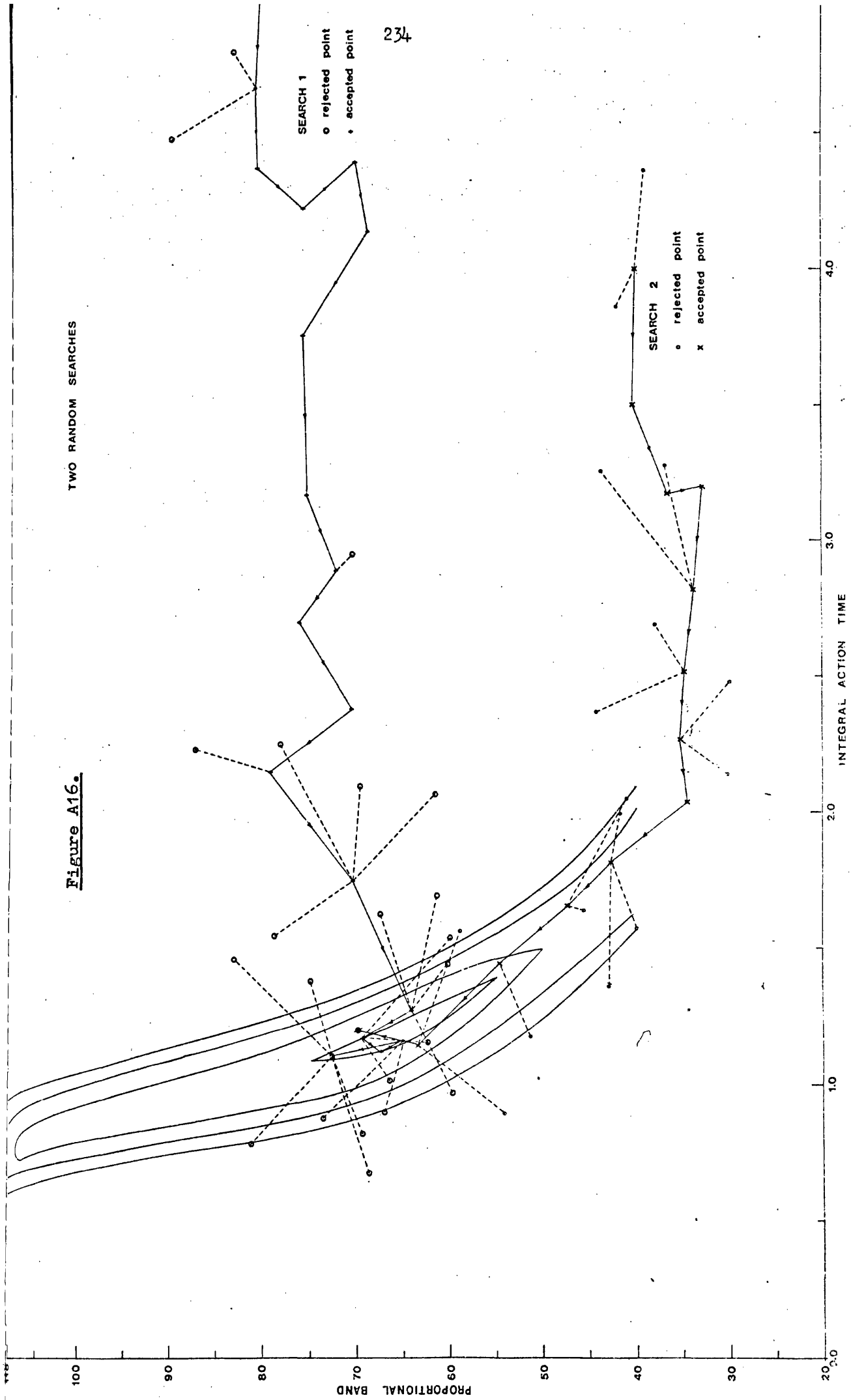
Figure A14. Continued

<u>Program Section</u>	<u>Function</u>
Lines 1.01 to 1.17	Initialization of system parameters
Lines 1.20 to 1.90	Timing control, sequencing and low set point error detection
Group 6	Data input and control
Group 8	Pre-program initialization of user scan flags
Group 9	Error integration routine
Group 11	Selection of operating point for next trial

Figure A15. Description of Random Search Program

TWO RANDOM SEARCHES

Figure A16.



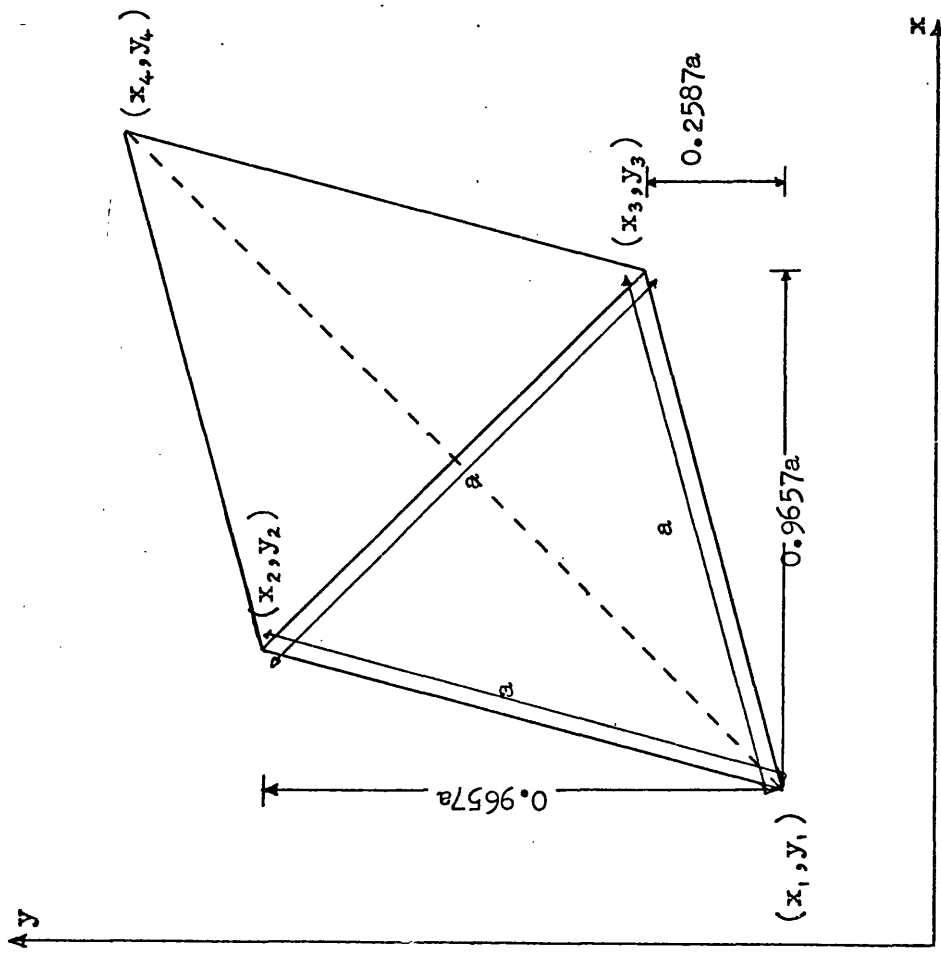


Figure A17. Starting Point Of a Simplex Search Procedure

APPENDIX B.

Effects of Errors in Timing upon the design
considerations of a Real-Time Operating System

B.1. Methods of Providing Synchronous Scanning Facilities

The structure of a Real-Time Operating System will be dependent to a large extent upon the accuracy required of a synchronous sampling scheme.

If a high degree of accuracy in synchronous timing was required, it would be imperative for all peripheral devices to be driven by the clock routines within the interrupt processor. This would require handlers for all peripheral devices to be available for use by the interrupt processor. The simple approach could be adopted, whereby all available devices were serviced, however this would lead to excessive timing overheads within the interrupt processor. It would therefore be necessary to allow users to specify which particular devices were to be used within his program so that only those device handlers actually required would be activated.

Input from and output to peripheral devices would have to be buffered in data tables which could be accessed by or loaded from commands available to the user in the high level language. The data table type structure is unfortunately inefficient with respect to core usage, as the same amount of core needs to be allocated irrespective of the number of peripheral devices being used. In a minicomputer environment where core space is very limited, peripheral device data tables would have to be restricted in length thereby reducing the number of devices which could be used.

In order to provide synchronous data processing facilities, which must be available for calculation of control outputs of various sorts, it would be necessary to provide software flags. These software flags would have to be accessible from commands or functions in the high level language so that a user would ensure that his data processing proceeded in synchronism with data input and output.

The above approach would ensure that errors in timing of input and output were reduced to a minimum level. However it would also impose severe limitations as to the number of peripheral devices which could be used and it would be preferable if a more flexible approach could be adopted.

An alternative to the above procedure would be to set a software clock flag within the interrupt processor which could be interrogated by a user from a command available within the high level language. If the flag was found in the set condition the user could then access the required peripheral devices from further commands available within the high level language.

This approach would avoid the necessity for using table driven peripheral devices thereby extending the number of peripheral devices which could be made available. It does however possess a severe disadvantage in that timing errors can occur.

B.2. Types of Timing Errors

A program sequence similar to that shown below would be necessary in order to set up a synchronous sampling scheme.

10.01	IF (FLAG (A))	10.10	
10.02	G 10.01		
10.10)	I/O tasks and
)	synchronous
)	computation
)	
10.mm	G 10.01)	Return to waiting
)	loop for next
)	sampling period

This would apply to either method but in the first method input output commands would merely be accessing data tables whereas in the latter method the commands would access peripheral devices directly.

Two types of timing errors could be incurred with such a sampling system:-

1. If the computational load within the sampling loop is too great, a complete sampling period could be omitted at some stage. This particular error can occur irrespective of the sampling procedure adopted.
2. As it takes a finite time to assess whether the timing flag is set a random error in sampling time will occur. The length of this particular sampling error will be between 0 seconds and the time taken for the execution of the two statements

within the synchronizing waiting loop. In the above example this could be as high as 10 m seconds.

In the case of all peripheral devices being driven from the interrupt processor this particular type of timing error will not occur as the flag is only used for synchronizing the processing of data. However in the alternative procedure where data input and output is also controlled by the flag, this error will always occur.

Fig.B.1 shows a diagrammatic representation of such a sampling scheme.

Before deciding upon which of the two sampling procedures to adopt, it was necessary to consider the effects caused by the above types of sampling errors.

B.2.1. Effects produced by missing a complete sampling period

It is possible to avoid this type of error by ensuring that the computational requirements of a sampling cycle do not exceed the period of the sampling cycle. This type of sampling error is therefore of little importance. However there is still a possibility of it occurring and it would be advantageous to know the effects it would have upon a system.

Dannenberg and Melsa (70) have studied the effects of missing a complete sampling cycle in a situation where a time shared computer is being used for control and data processing. They concluded that it is possible to occasionally skip a sampling cycle with little detriment to the stability of the system.

B.2.2. Effects produced by random errors in sampling times. "Jitter Sampling"

This type of process has been studied by AKAIKE (71) in order to determine the effects of such timing errors on the Power spectra of sampled signals.

For the process shown in Fig.B.1. where the sampling time T_n is related to the sampling period Δt by the equation

$$T_n = n\Delta t + e_n$$

Where e_n is the error in sampling time at the n^{th} sampling instant and is one of the set of independent random variables from the same probability distribution.

AKAIKE has established the following relationship between the true and aliased power spectra of a jitter sampled signal

$$P_t(f) = |\Phi(f)|_A^2 \cdot P_A(f) + \Delta t \cdot \int_{-\infty}^{\infty} (1 - |\Phi(f')|^2) P(f') df'$$

where

$P_t(f)$ is the true power spectra of the process

$P_A(f)$ is the aliased power spectra caused by periodic sampling (B3)

$\Phi(f)$ is the characteristic function of the probability density function $g(e)$ of the deviation in sampling intervals

$$\Phi(f) = \int_{-\infty}^{\infty} \exp(2\pi i f e) \cdot g(e) \cdot de$$

and $|\Phi(f)|_A^2$ is the aliased version of $|\Phi(f)|^2$

According to AKAIKE this can be interpreted as a white noise source of magnitude $\int_{-\infty}^{\infty} (1 - |\Phi(f')|^2) \cdot P(f') \cdot df'$ with a filtered version of the original process, the transfer function of the filter being $|\Phi(f)|_A^2$

For no timing errors, the relationship reduces to

$$P_t(f) = P_A(f)$$

indicating that the sampling process causes aliasing of the original $x(t)$ signal.

When timing errors occur

$$|\Phi(f)|^2 < 1 \quad \text{for } f \neq 0$$

$$\lim_{f \rightarrow \infty} |\Phi(f)|^2 = 0$$

Thus concluding that this particular time sampling process acts as a low pass filter, the power which was present in the higher frequency components of the signal being transferred into the white noise effects.

The effects of jitter in sampling intervals is therefore to distort any high frequency components of a signal and can therefore be minimized by selecting a sampling rate which is significantly greater than the highest frequency component present in the signal.

B3. Simulation Studies

A simulation study was carried out so as to obtain quantitative data on the effects that jitter sampling would have on signal recovery and whether it would make the use of an interrupt driven system essential. Simulation programs were written in FORTRAN IV and compiled and executed on an I.C.L. system 4.50.

The methods adopted involved the generation of data as if a sinusoidal signal had been sampled with jitter sampling procedure and then to analyse the data.

- a. by attempting to recover the signal from the sampled data using a least squares fitting technique of a simple model and assuming that the data came from an equispaced sampling procedure.
- b. by evaluating and examining the power spectrum of the sampled data using a method for equispaced data.

B.3.1. Data Generation

In the type of sampling system envisaged, the actual sampling time will always be later than or the same as the expected sampling time. The deviation between actual and expected sampling times will vary between 0 and some maximum value with all values being equally likely. Fig.B.2.

To simplify matters, it was decided that the original continuous signal should be a single sinusoidal function of the form.

$$y = \sin (W_0 t)$$

where W_0 is the angular frequency of the signal
sampled data was generated in the form

$$y_n = \sin W_0 (t_n + e_n)$$

where:-

t_n is the expected sampling time

e_n is the deviation between actual and expected sampling time and is of the form

$$e_n = m r_n$$

where r_n is a random number in the range 0 to 1, the probability distribution function of the r_n s was uniform so that all numbers in the range 0 to 1 were equally likely.

m is a scaling factor so that the maximum value of the random deviation may be varied as a fraction of the nominal sampling period

If a nominal sample period of 1 second is chosen, data sets can be generated as if sampling at various rates by altering the angular frequency W_0 of the input signal.

Sampling rates are most conveniently expressed in terms of multiples of the Nyquist rate, this being the minimum rate at which samples can be taken so as to be able to obtain full signal recovery.

The Nyquist Rate $N \equiv 2$ samples per cycle. At a sampling rate of 1 sample per second, the maximum signal frequency must be $\frac{1}{2}$ cycle per second $\equiv \pi$ radians per second.

maximum value of $W = \pi$ radian / sample

B.3.2. Least Squares Analysis of Data

Assuming that the sampled data produced would have the same nominal angular frequency as the original signal, the model

$$y = a_0 + a_1 \sin W_0 t + a_2 \cos W_0 t \quad \dots (1)$$

was used in a least squares curve fitting technique so as to examine the amplitude and phase distortions produced by the sampling.

The least squares technique involves the solution of the equation

$$\begin{aligned} a_0 n &+ a_1 \sum_{i=1}^n \sin W_0 t_n + a_2 \sum_{i=1}^n \cos W_0 t_n = \sum_{i=1}^n Y_n \\ a_0 \sum_{i=1}^n \sin W_0 t_n + a_1 \sum_{i=1}^n \sin^2 W_0 t_n + a_2 \sum_{i=1}^n \sin W_0 t_n \cdot \cos W_0 t_n &= \sum_{i=1}^n Y_n \sin W_0 t_n \\ a_0 \sum_{i=1}^n \cos W_0 t_n + a_1 \sum_{i=1}^n \sin W_0 t_n \cdot \cos W_0 t_n + a_2 \sum_{i=1}^n \cos^2 W_0 t_n &= \sum_{i=1}^n Y_n \cos W_0 t_n \end{aligned}$$

in order to determine the values of the constants a_0 a_1 a_2 in the model equation.

Equation (1) can be rearranged to give

$$y = a_0 + b \sin (W_0 t + \phi) \quad \dots (3)$$

where a_0 is the zero drift of the sampled signal

$$\begin{aligned} b &\text{ is the amplitude of the reconstructed signal} \\ &= \sqrt{a_1^2 + a_2^2} \quad \dots (4) \end{aligned}$$

is the difference in phase between the recovered signal
and the original sinusoidal function

$$= \tan^{-1} \left[\frac{a_2}{a_1} \right] \quad \text{.. (5)}$$

The results obtained can therefore be conveniently represented in terms of plots of amplitude recovery and phase difference against sampling frequency. This method will give some indication of the distortion of the signal caused by the jittered sampling process provided of course that the results obtained are statistically significant.

As all the sampling times never vary the data should approximate to the equation:

$$y = \sin \omega_0 \left[t + \frac{m \Delta t}{2} \right] \quad \text{.. (6)}$$

where t is the time interval between samples and m is maximum fraction of the sampling period by which the sample will be delayed.

Figures B.3 to B.12 show the results of the computer simulator study for a data set of 100 points. The results are plotted as amplitude against sampling rate and phase difference against sampling rate for deviation in sampling time of between 0.2 and 1 sampling period as labelled.

From the results of this simple example it can be observed that the distortion of the recovered signal with respect to the input signal increases as the maximum deviation in sampling time approaches a whole sampling interval. Furthermore, when the sampling rate is increased with respect to the signal frequency, the distortion is reduced.

The continuous line on the graphs of phase difference vs sampling rate is that of the phase lag predicted from equation (6), and indicates that the recovered signal could be represented adequately by just a phase lagged version of the original signal.

B.3.3. Spectral Analysis of Data

Estimates of the Power spectrum of the sampled data were made using the method outlined by Blackman and Tukey (72, 73).

Assuming that the data is available at equispaced intervals of time estimation of the auto correlation function can be made at the times for which the value of the function is known.

i.e. For a sampling interval of Δt data is available at

$$t = 0, \Delta t, 2\Delta t, \dots, n\Delta t$$

and the auto correlation function at these values of time

$$C_r = \frac{1}{n+1-|r|} \cdot \sum_{s=0}^{n-r} (X_{r+s} - \bar{X}) \cdot (X_s - \bar{X})$$

for $0 \leq r \leq m$ where m is generally taken as about 10% to 15% of the maximum number of values within the data set.

\bar{X} is the mean of the data set X_0 to X_n

A crude estimate of the power spectrum of the data can be made by using the unconvoluted cosine transfer.

$$V_q = C_0 + \sum_{r=1}^{m-1} 2C_r \cos \frac{qr\pi}{m} + C_m \cos q\pi$$

for $q = 0, 1, \dots, m$

Finally, smoothed estimates of the power spectrum can be obtained from the crude estimates by using the Hamming Spectral window

$$U_0 = 0.54.V_0 + 0.46.V_1$$

for $q = 1, 2, \dots, m-1$

$$U_q = 0.23.V_{q-1} + 0.54.V_q + 0.23.V_{q+1}$$

$$U_m = 0.46.V_{m-1} + 0.54.V_m$$

The resulting values of U_q are obtained for frequencies near $\frac{q}{2M}$ cycles per observation.

As only a single sided transfer has been considered, i.e. for positive time and frequencies only, the results obtained should be doubled. However, it is only the relative magnitude of the values which is of any concern and it is not really necessary to do this.

Figs. B.13 - 17 show the power spectral estimates obtained from a generated data set of 800 points, sampled at different sampling rates. Each diagram shows the variation produced as the deviation in sampling rate is increased from zero up to a whole sampling rate.

Figs.B.18 - 22 show power spectral estimation obtained by generating data in the form

$$Y_n = \sin W_o (t_n + e_n) + \sin \frac{W_o}{2} (t_n + e_n)$$

These tended to show that the higher frequency component was distorted to a greater extent than the lower frequency component.

B.4. Results and Conclusions

The results obtained by both methods confirm the theoretical analysis, high frequency components of the signal being distorted for more than the lower frequency components. (Frequency in this context must be compared with the sampling frequency). The distortion is therefore decreased as the maximum error in sampling time decreases and also as the relative frequency of sampling is increased.

It would appear that for signal recovery therefore that absolute sampling synchronism is not essential. For a sampling frequency of five times the minimum Nyquist rate a deviation value of about one-third of a sampling interval would be perfectly acceptable and a deviation of up to one-half of the sampling interval could be tolerated.

In an interpretive system where commands take of the order of 5 to 10 m seconds each, a free running sampling system would require approximately 20 or 30 m seconds per sample, i.e. to pick up data, perform a simple scaling operation and store the data. By including a software clockflag for synchronism, the time taken per data sample would be extended to approximate 50 m seconds with a maximum delay of about 10 m seconds.

Therefore it would appear that by using a slower interpretive type system, it would not be essential to have a totally interrupt driven system. Synchronous sampling can be accomplished simply and effectively by allowing users to interrogate a clockflag and then access the peripheral device when the flag is found in the set condition. The necessity for table driven software can therefore be avoided.

This view is also confirmed when one considers the scanning rates which are likely to be used in most applications (B.5) and also as explained in Chapter 4 high data capture rates can be accomplished in a different manner.

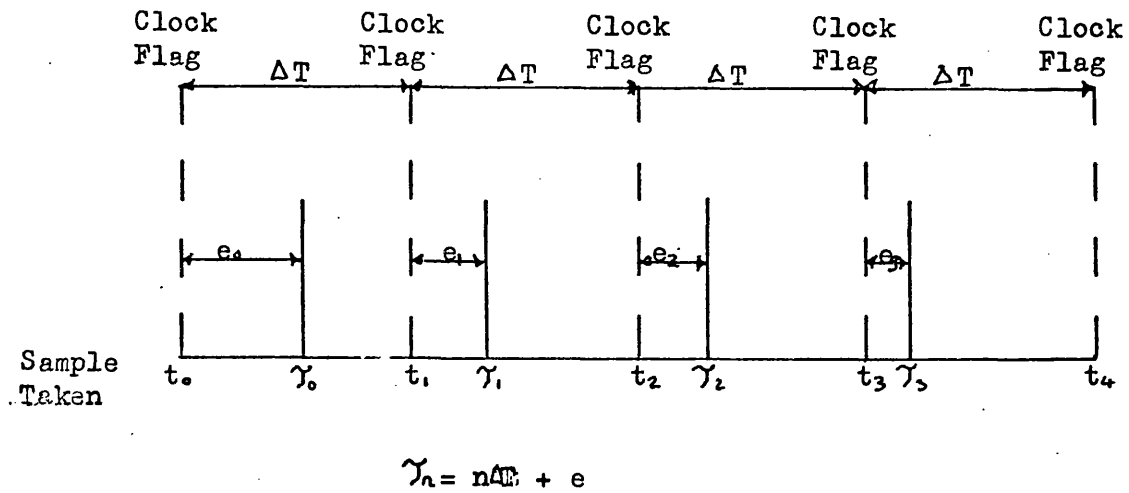


Figure B1. Expected Sampling Scheme

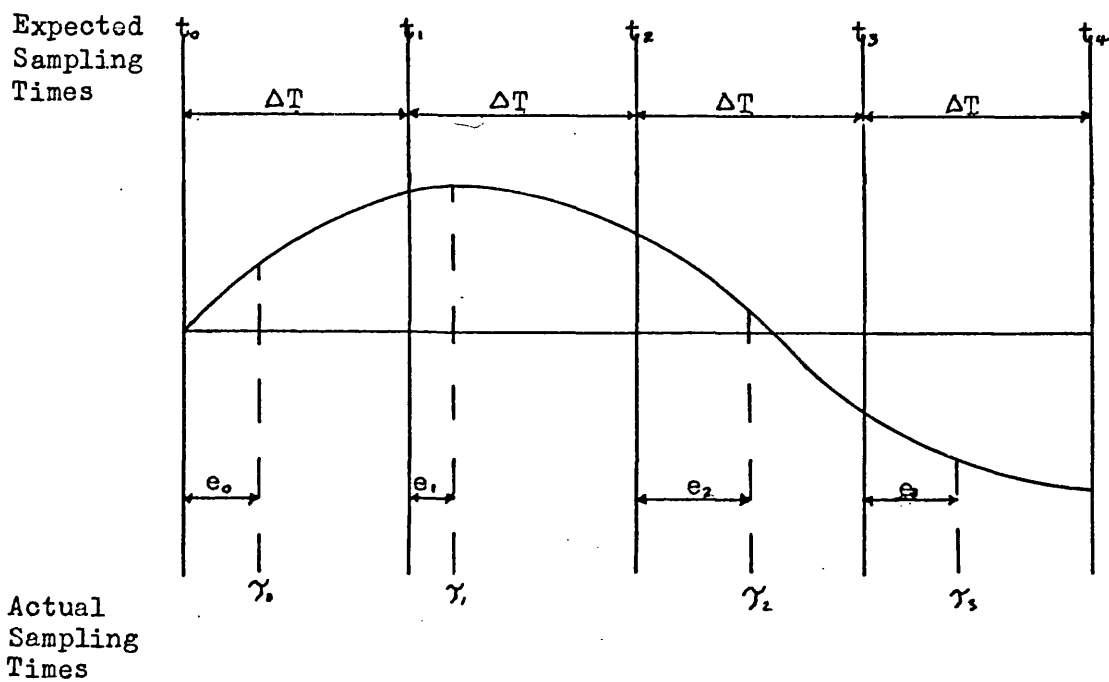


Figure B2. Signal Sampling with Jitter

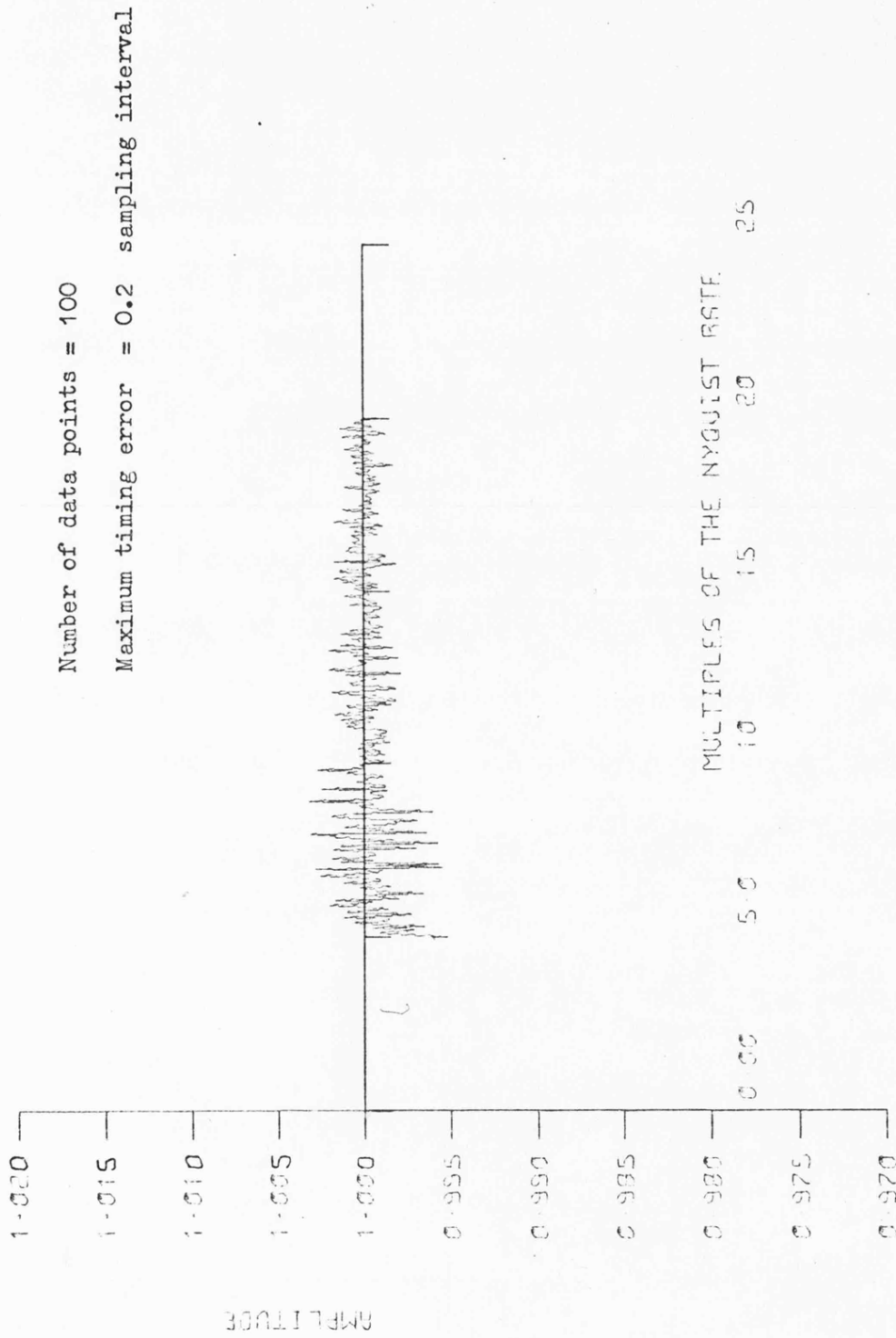
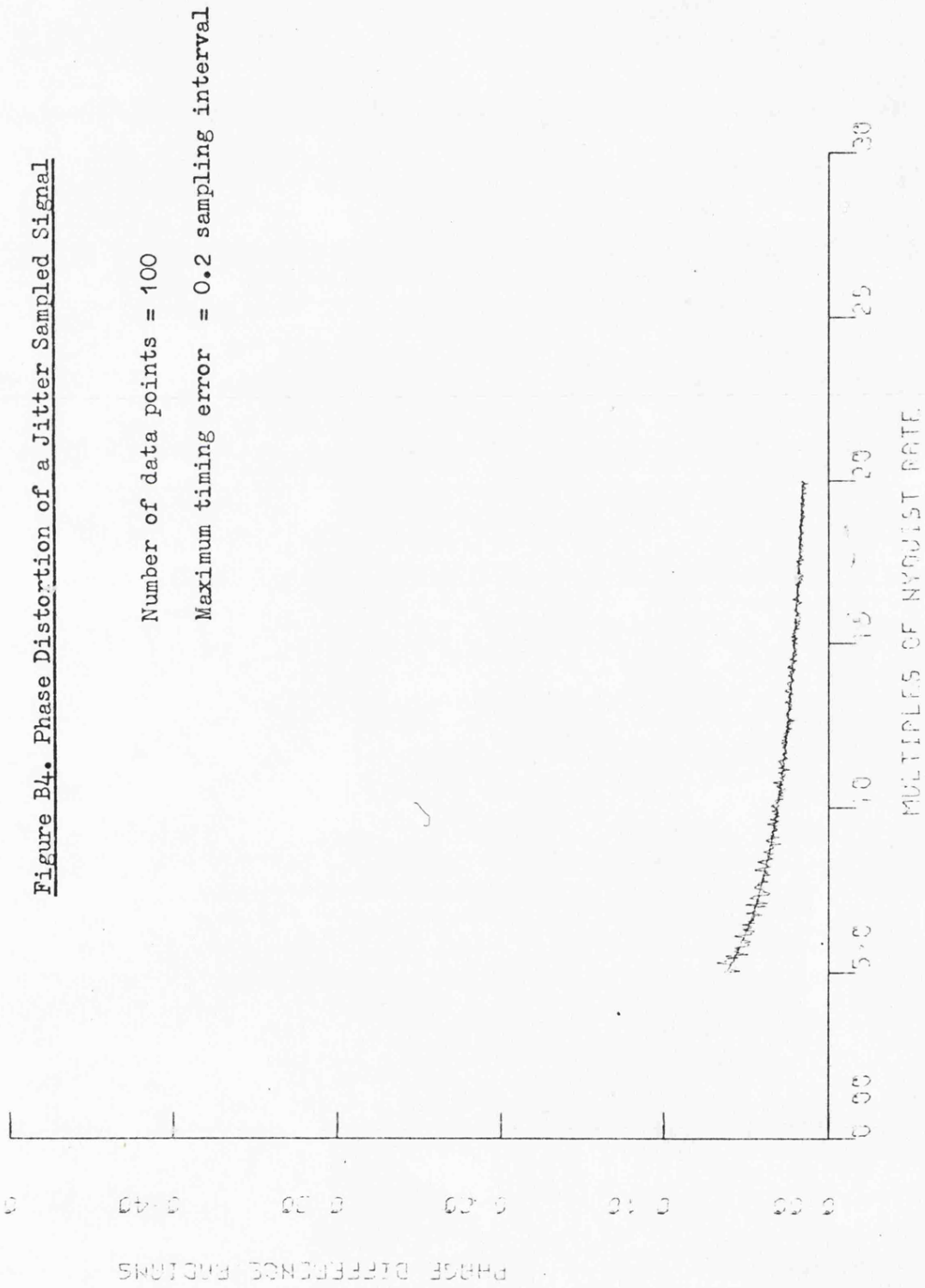


Figure B3. Amplitude Distortion of a Jitter Sampled Signal

Figure B4. Phase Distortion of a Jitter Sampled Signal



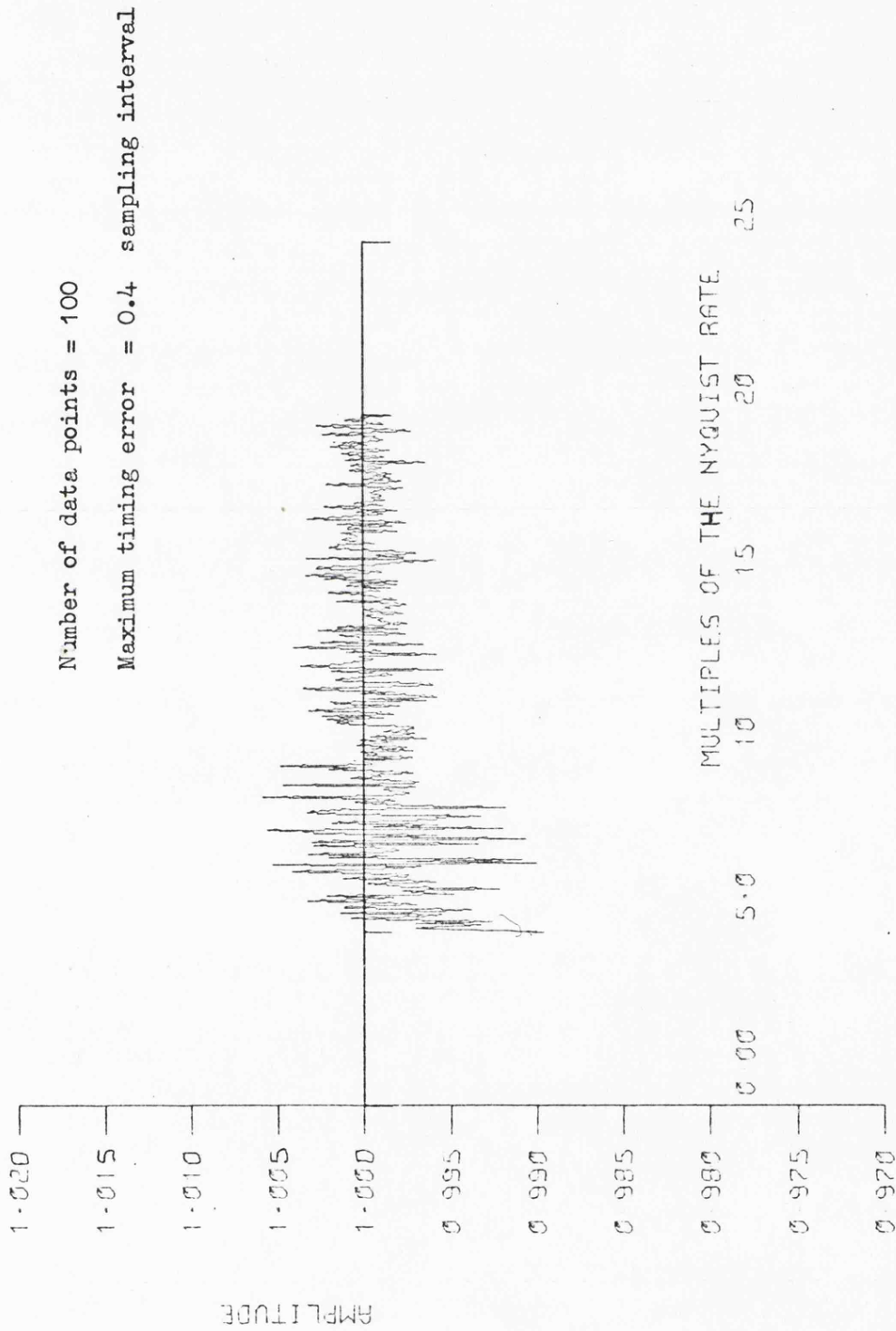
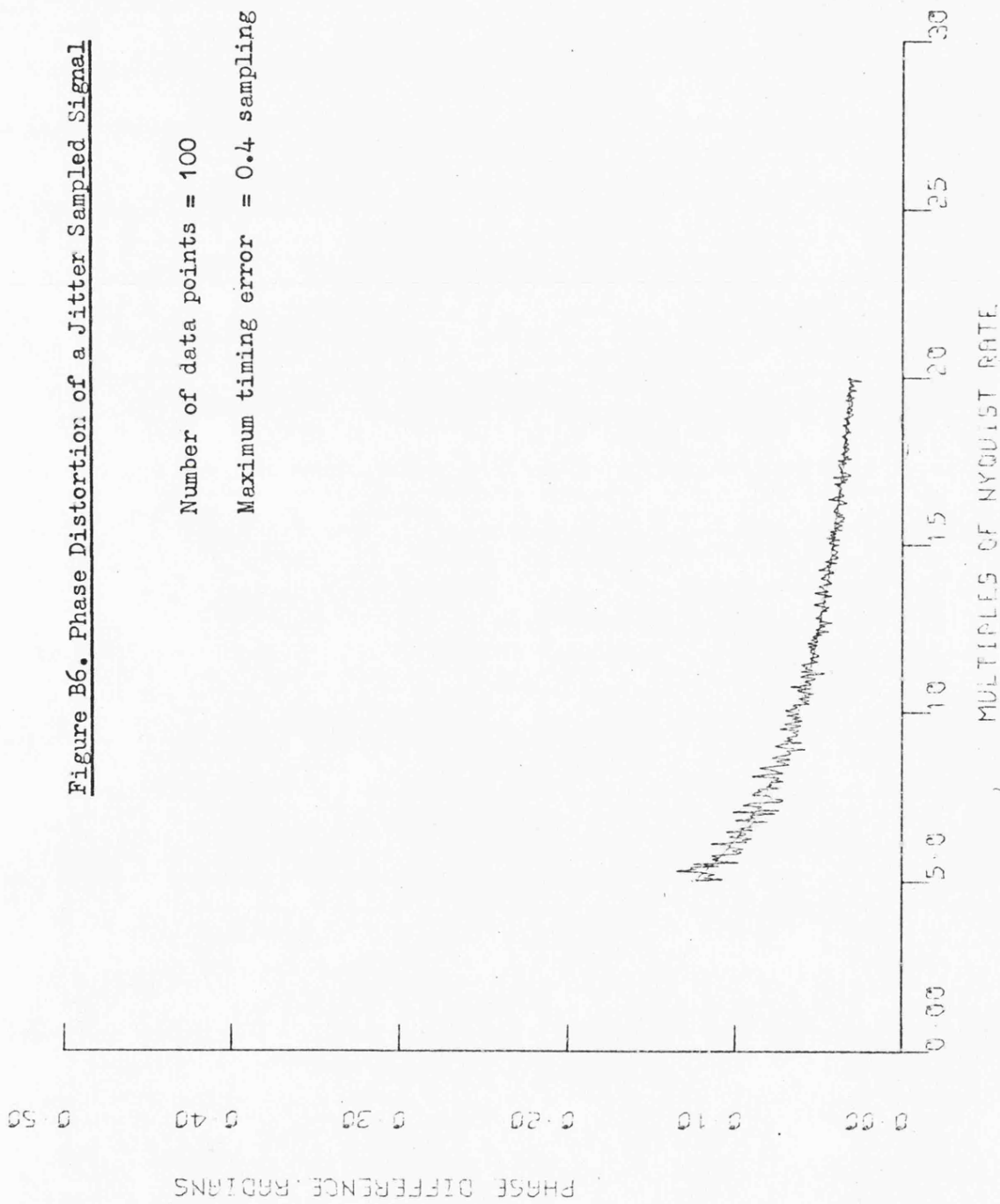


Figure B5. Amplitude Distortion of a Jitter Sampled Signal

Figure B6. Phase Distortion of a Jitter Sampled Signal

Number of data points = 100
 Maximum timing error = 0.4 sampling interval



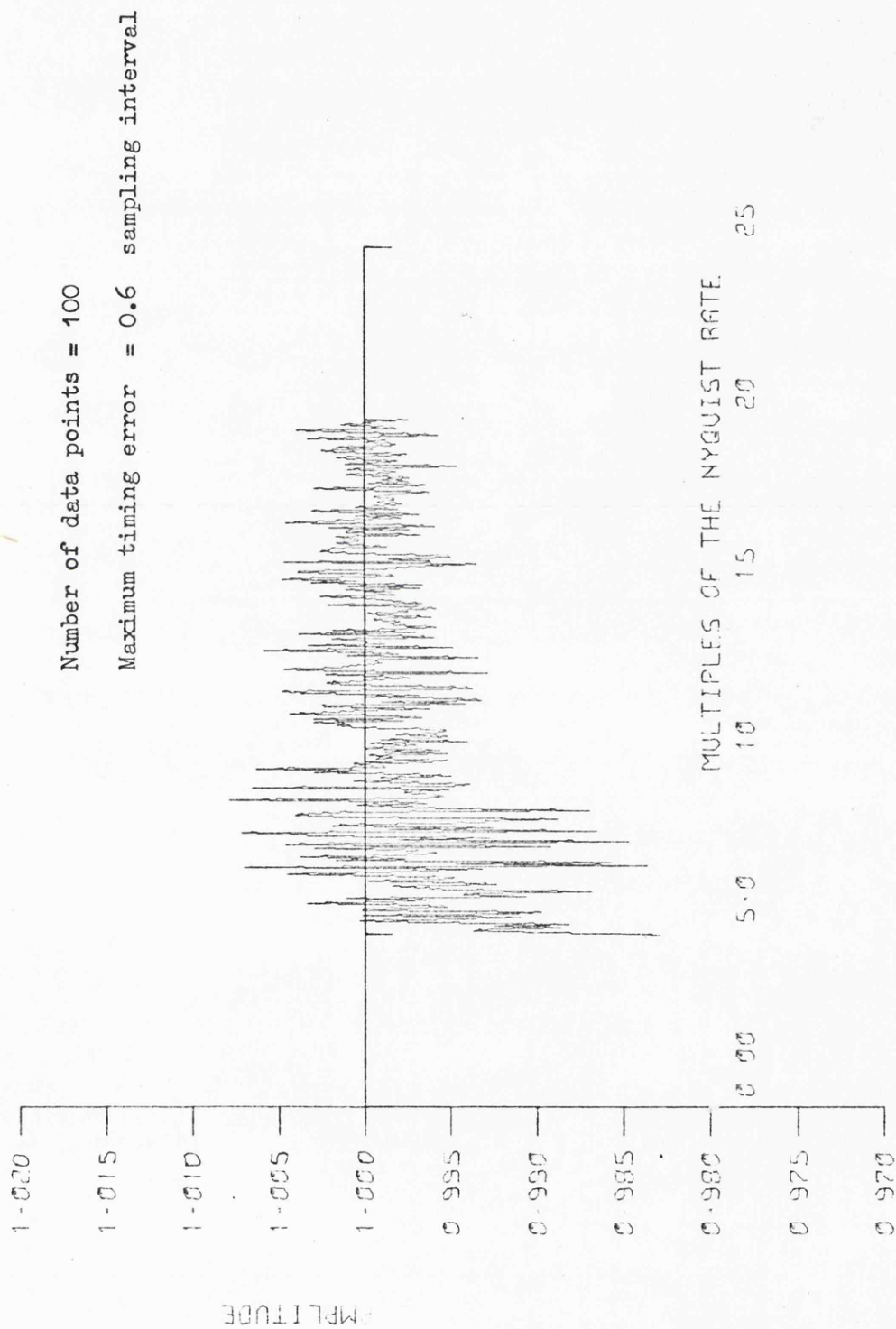
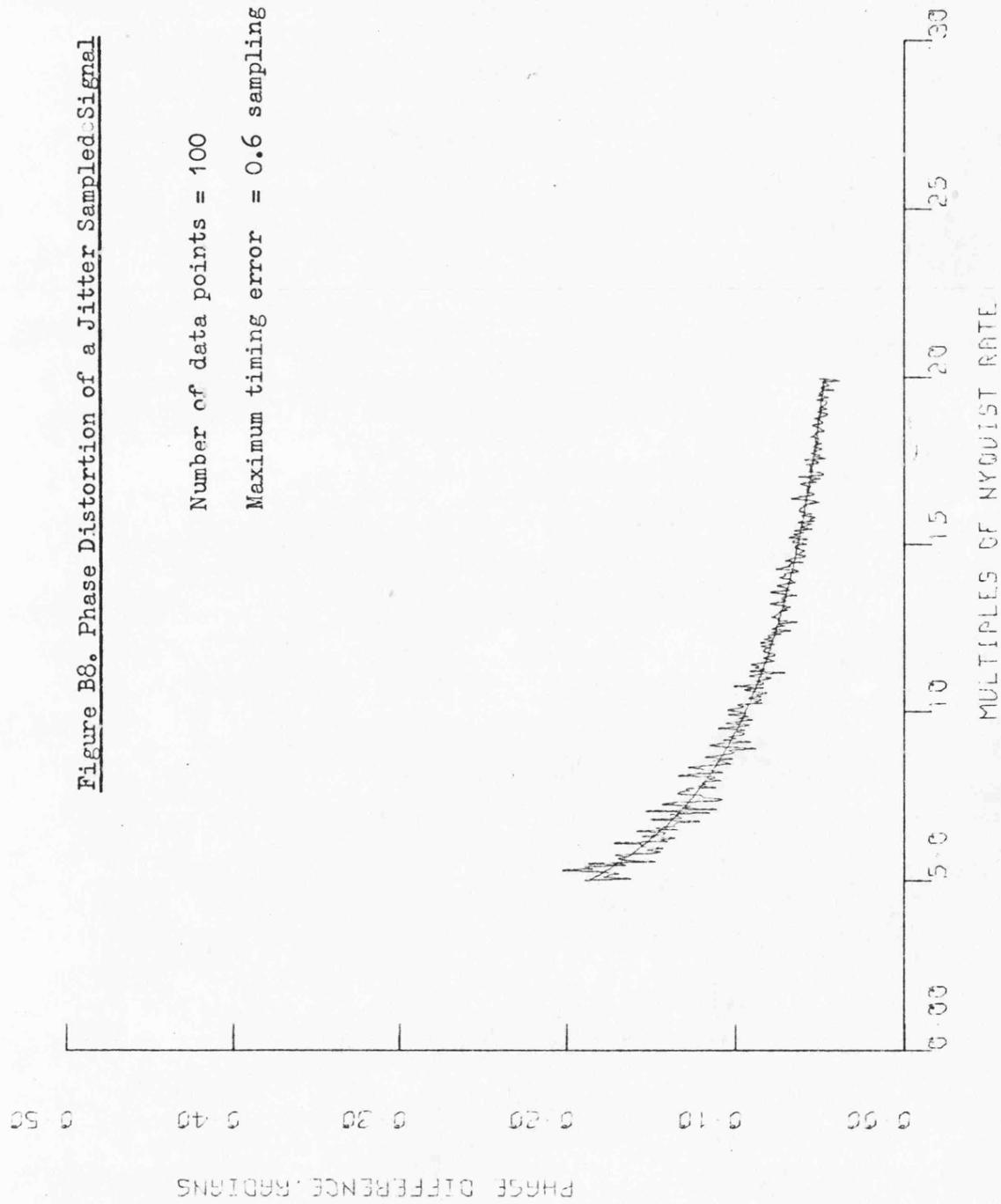


Figure B7. Amplitude Distortion of a Jitter Sampled Signal

Figure B8. Phase Distortion of a Jitter Sampled Signal

Number of data points = 100
Maximum timing error = 0.6 sampling interval



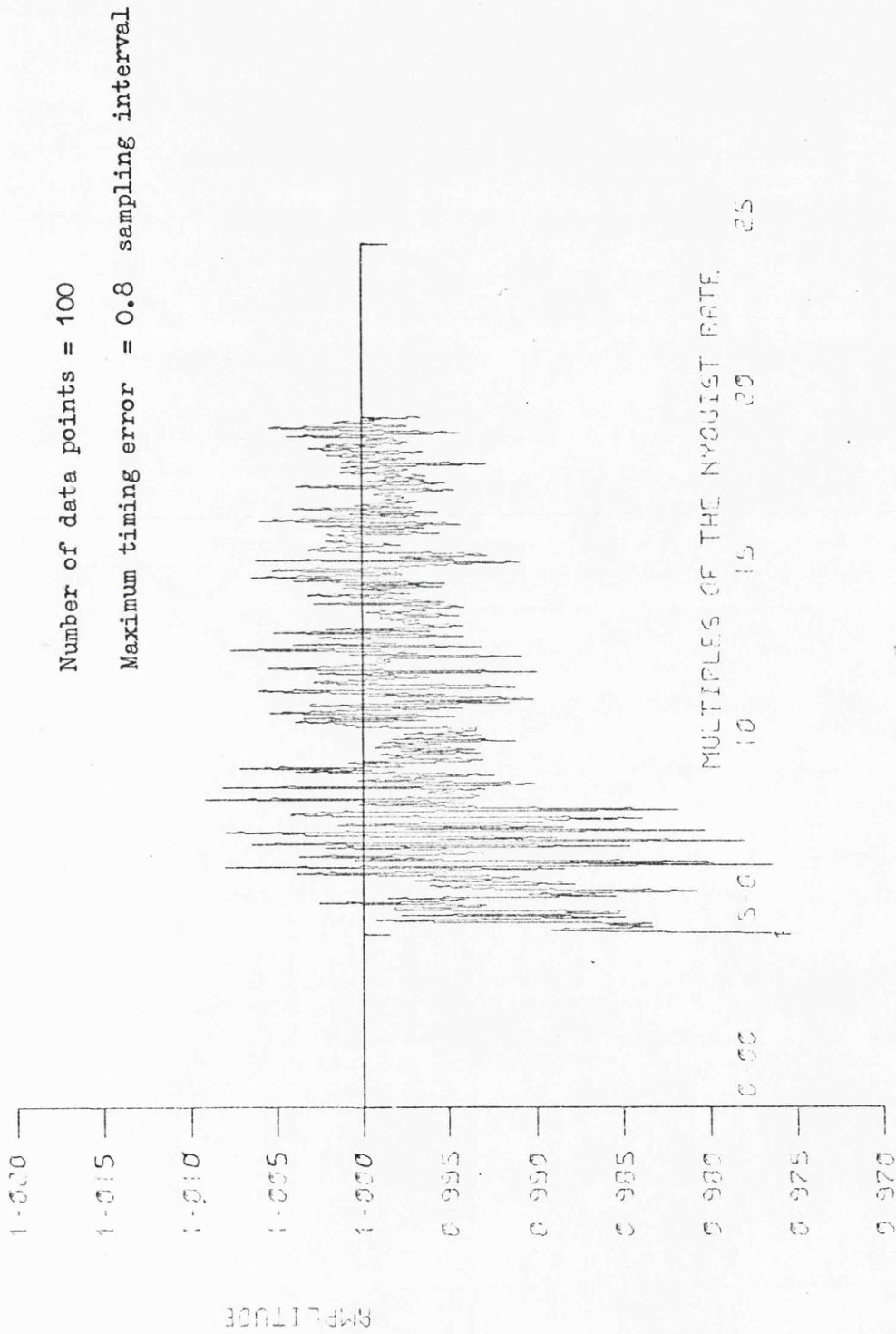
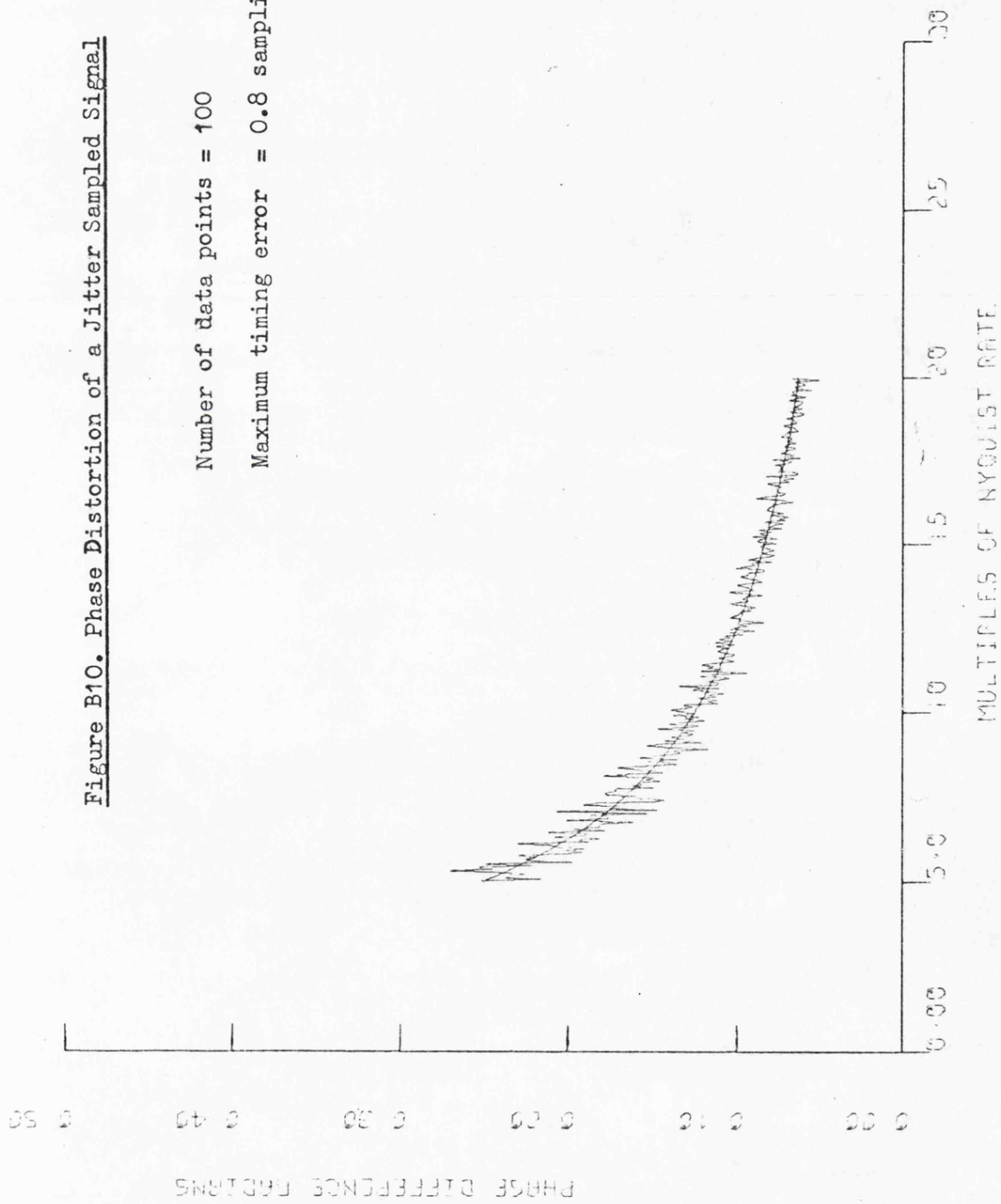


Figure B9. Amplitude Distortion of a Jitter Sampled Signal

Figure B10. Phase Distortion of a Jitter Sampled Signal

Number of data points = 100

Maximum timing error = 0.8 sampling interval



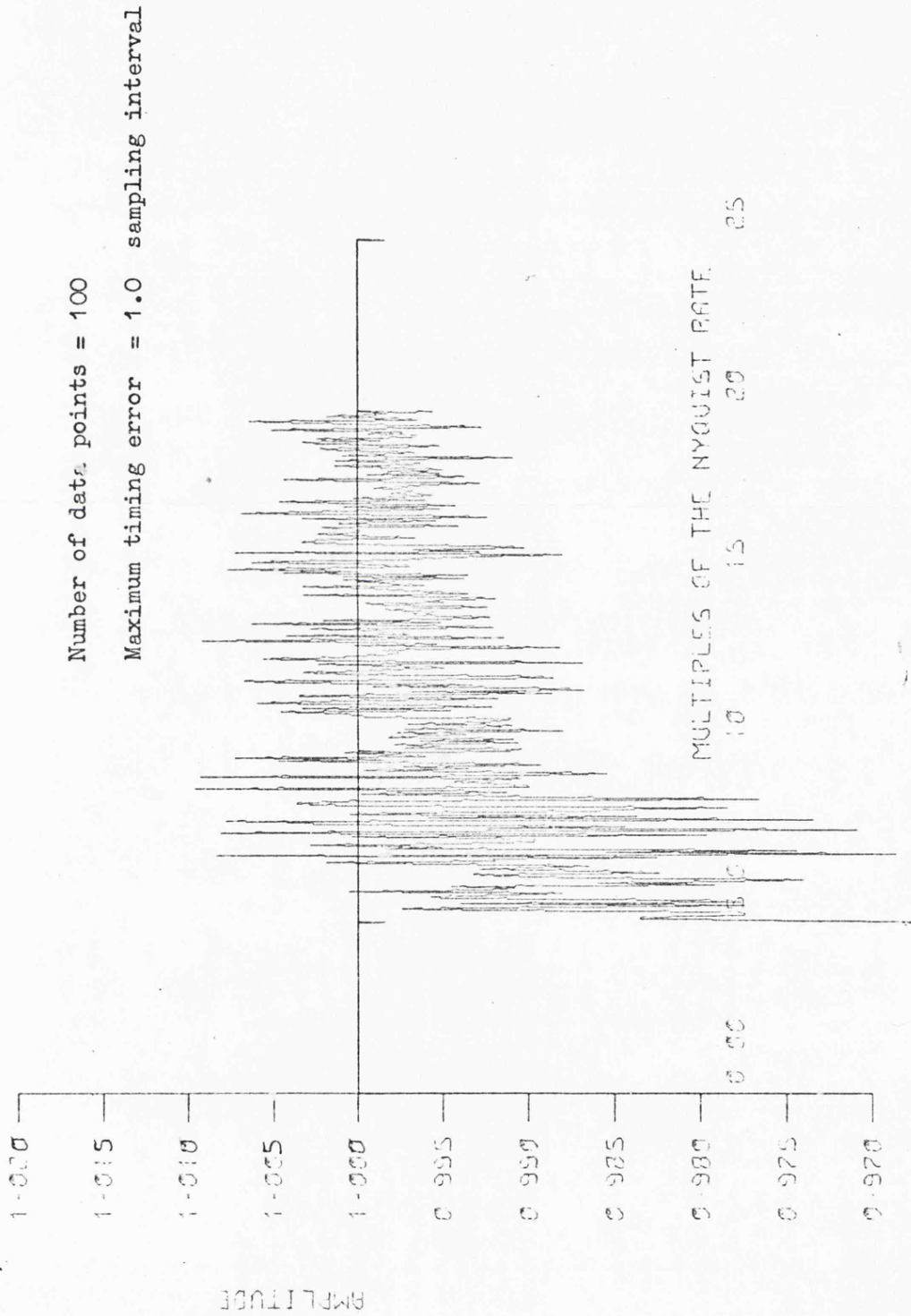
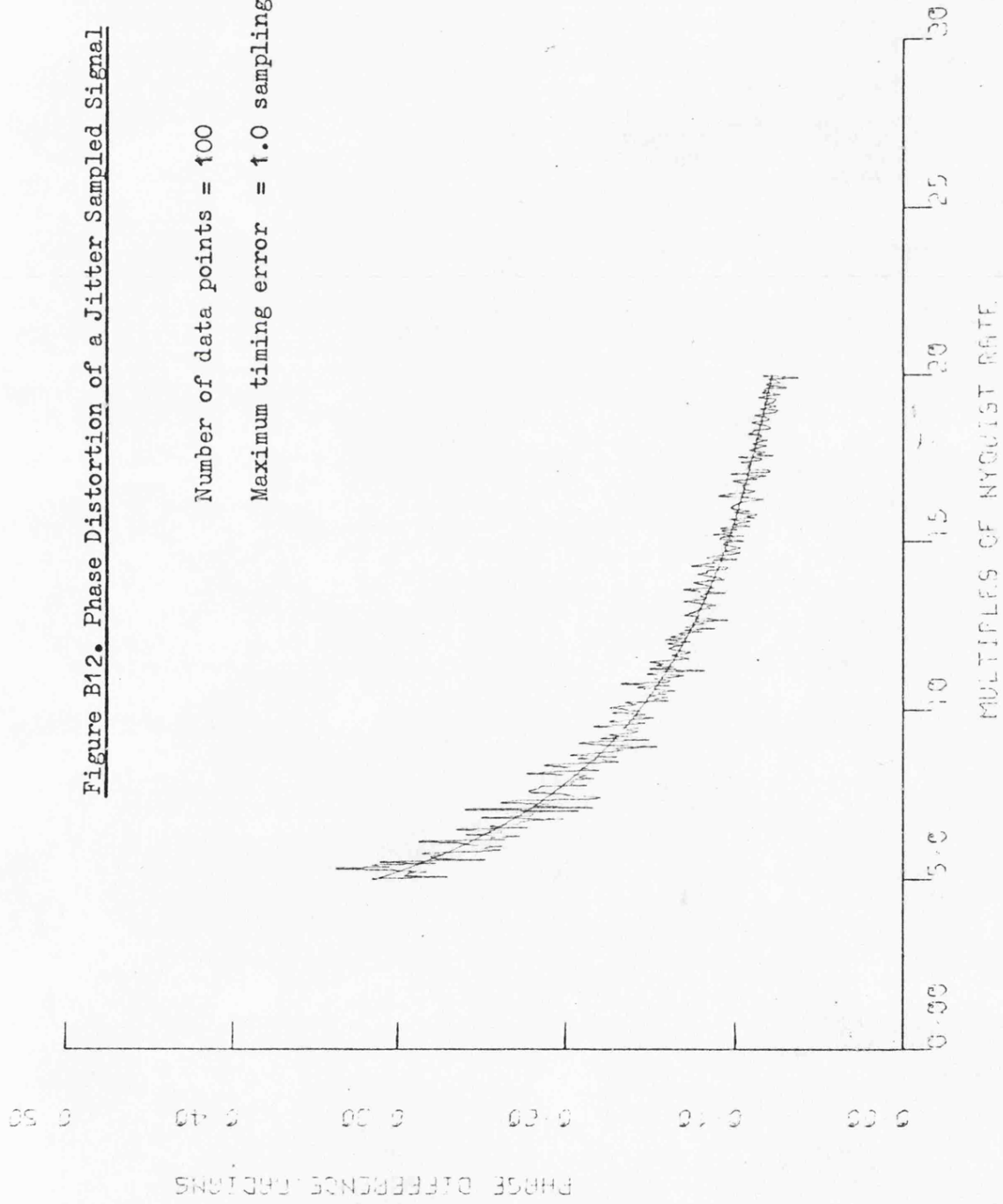


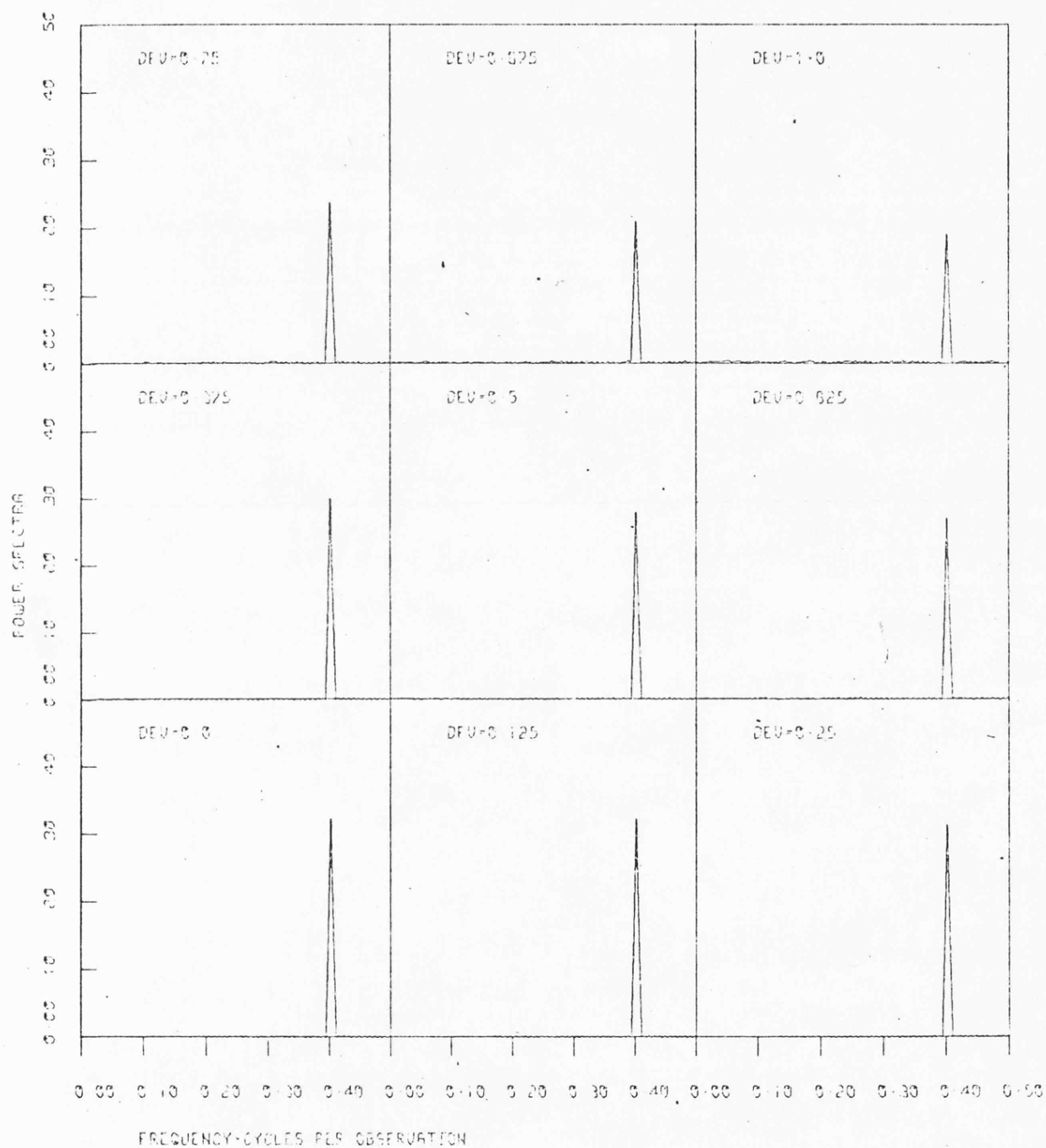
Figure B41. Amplitude Distortion of a Jitter Sampled Signal

Figure B12. Phase Distortion of a Jitter Sampled Signal

Number of data points = 100

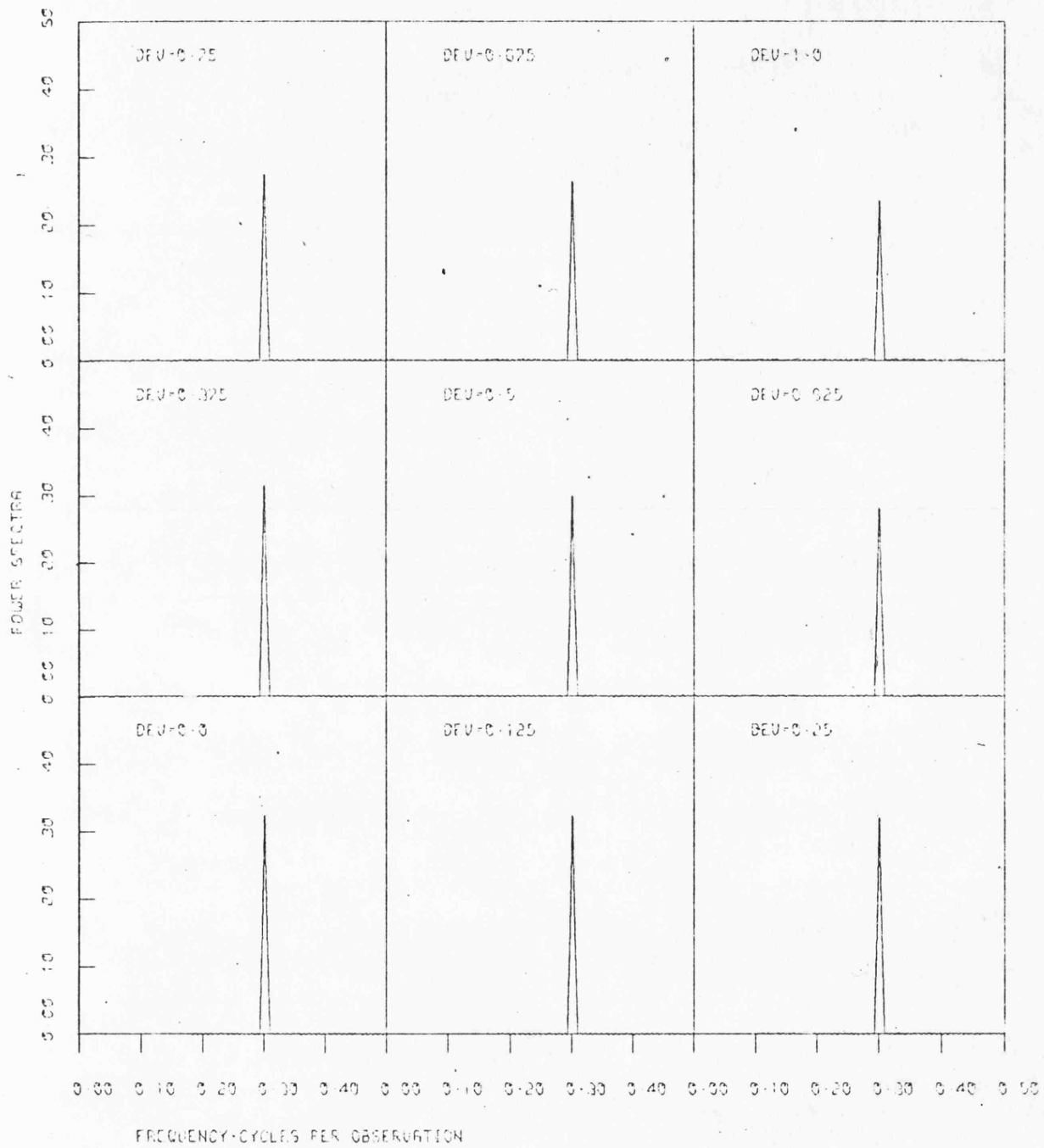
Maximum timing error = 1.0 sampling interval





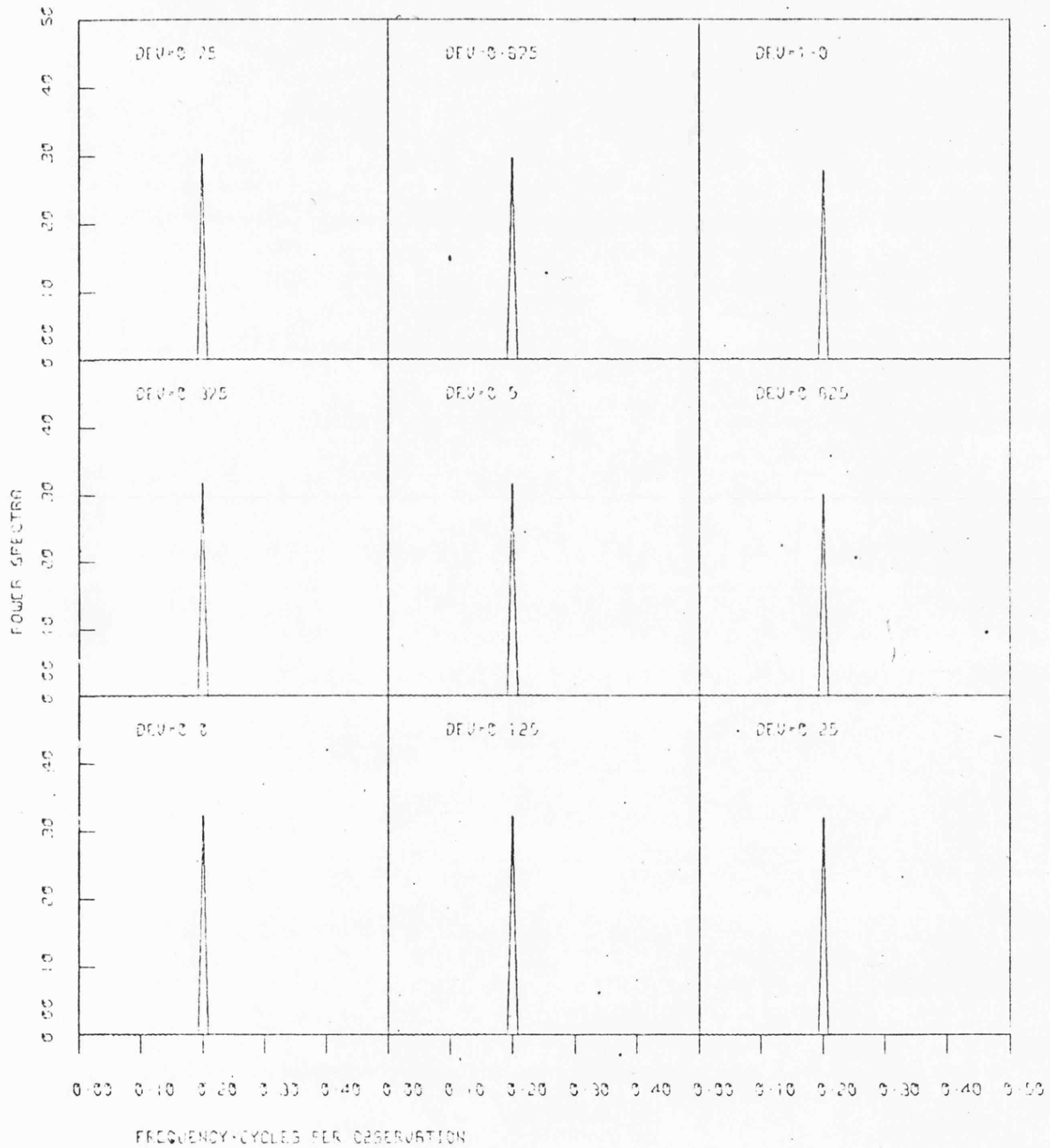
SPECTRAL ANALYSIS

Figure B13. 800 Data Points Sampled at 1.25 times Minimum Nyquist Rate



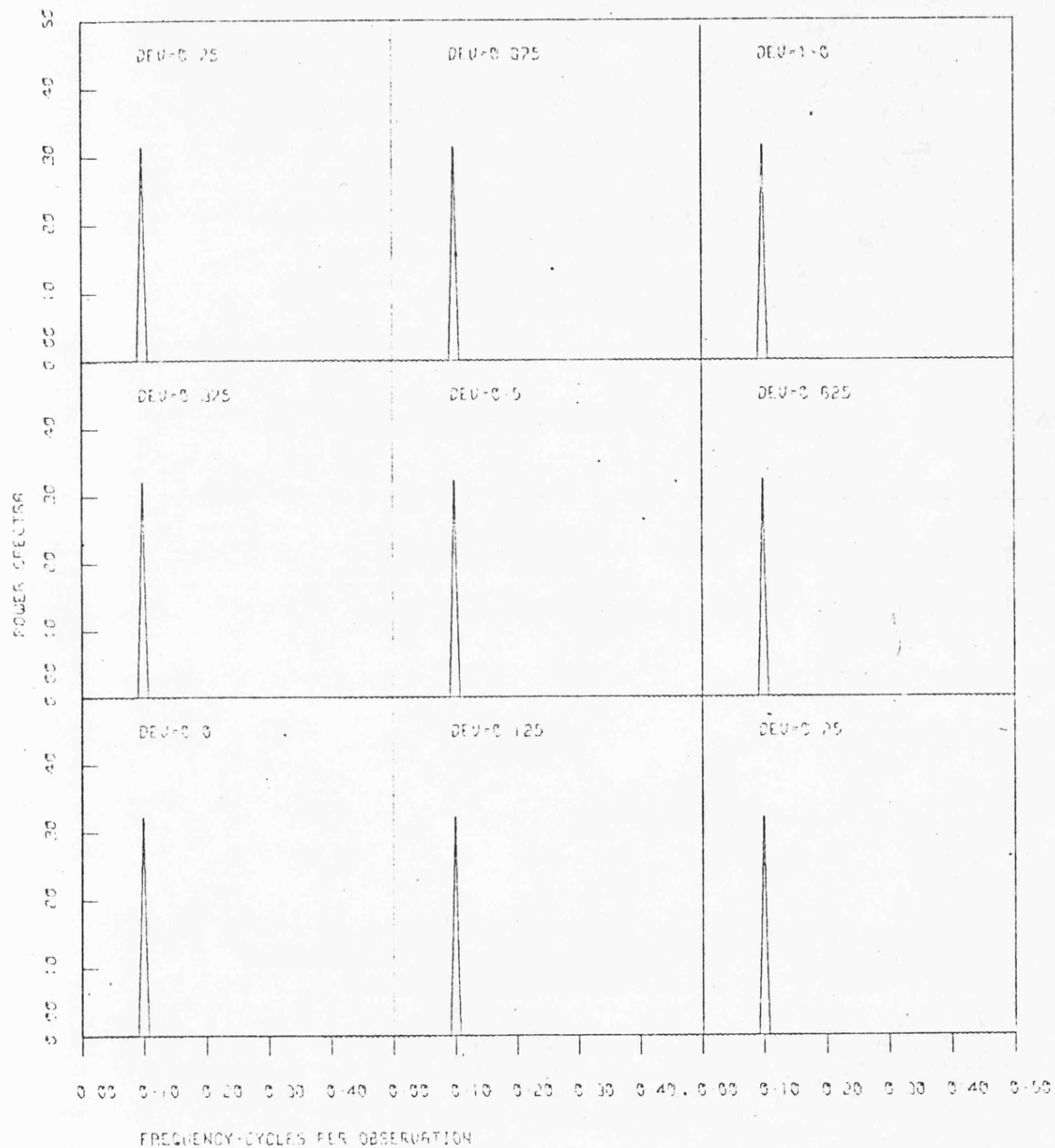
SPECTRAL ANALYSIS

Figure B14. 800 Data Points Sampled at 1.66 times Minimum Nyquist Rate



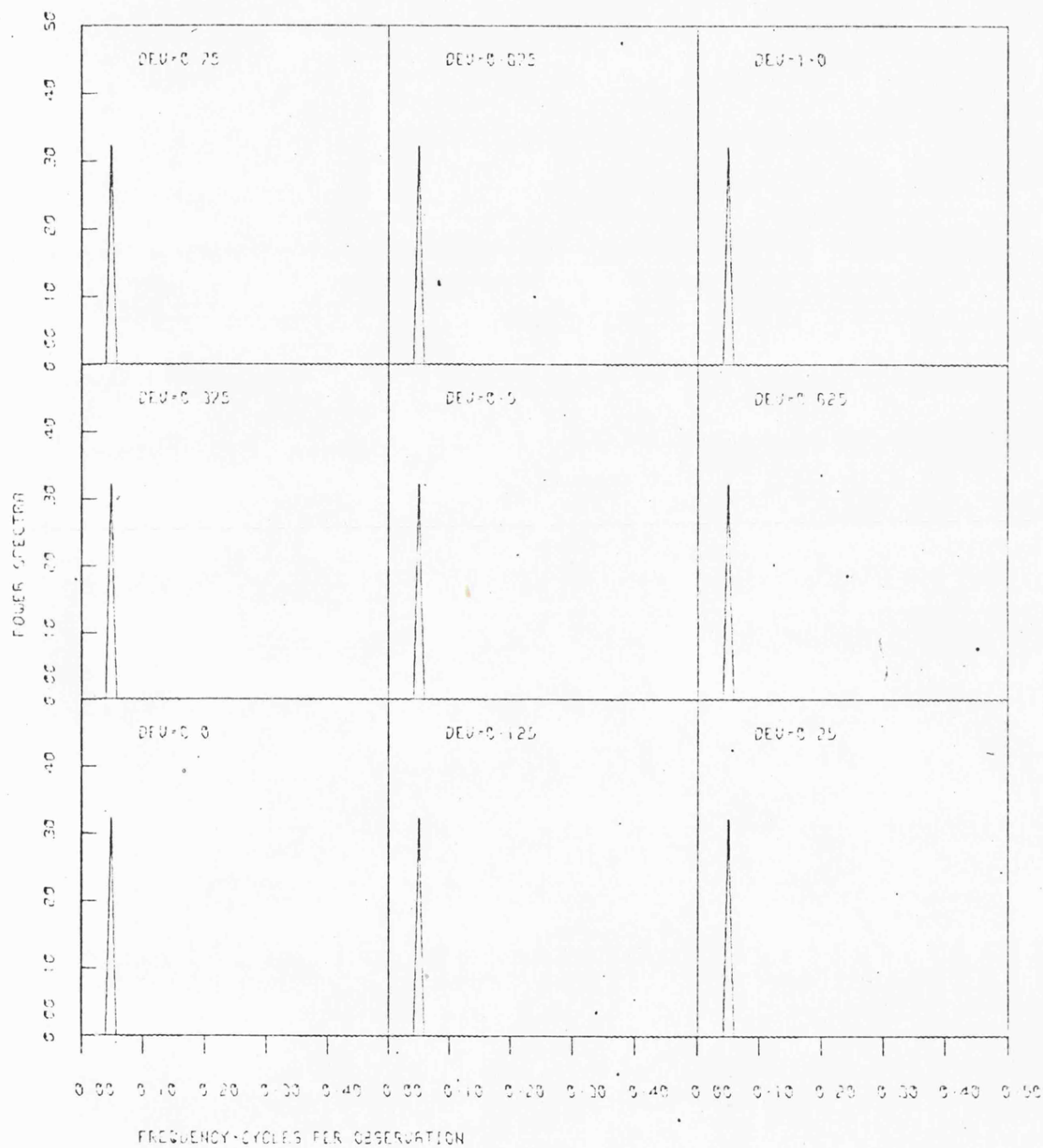
SPECTRAL ANALYSIS

Figure B15. 800 Data Points Sampled at 2.5 times Minimum Nyquist Rate



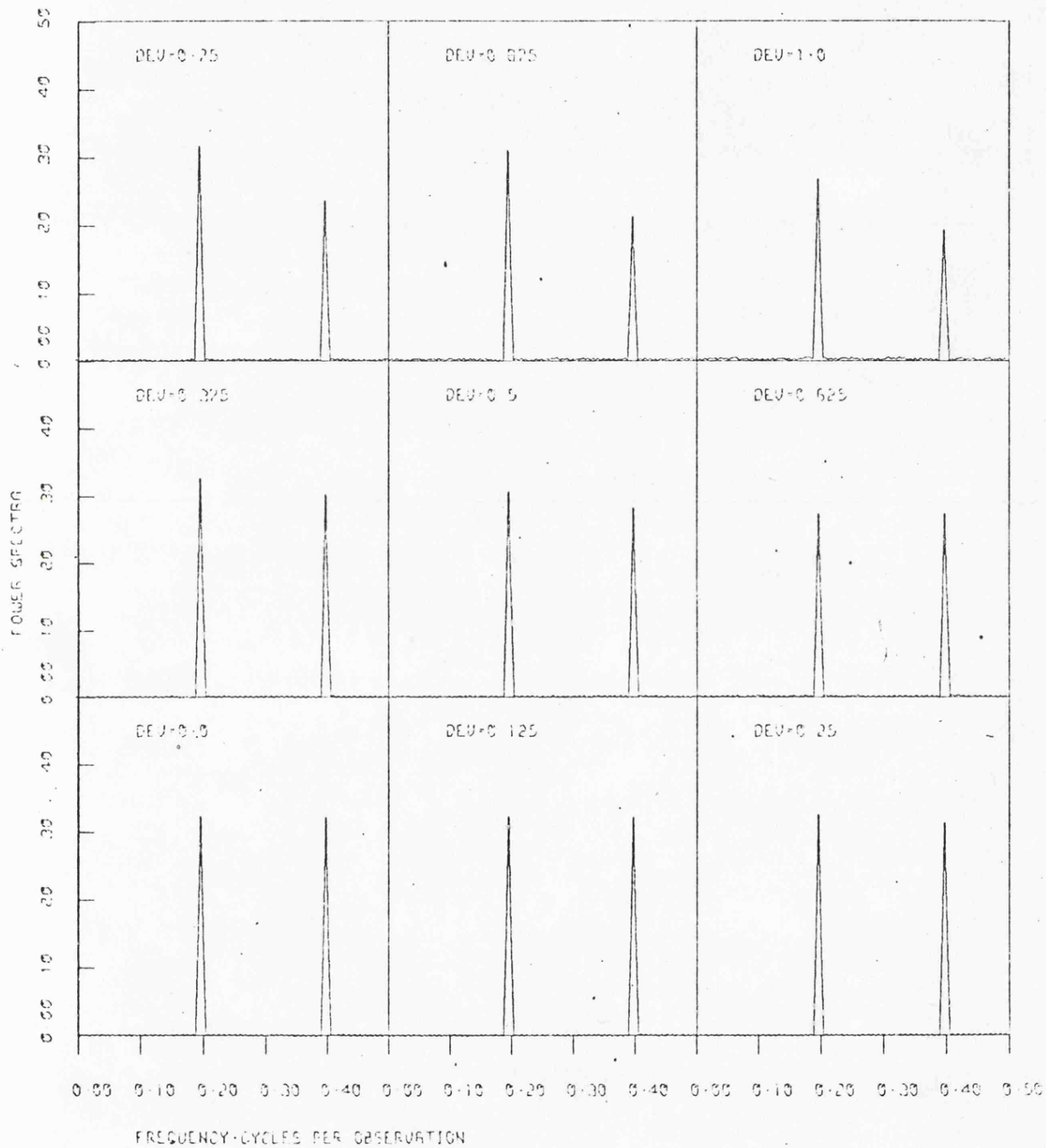
SPECTRAL ANALYSIS

Figure B16. 800 Data Points Sampled at 5.0 times Minimum Nyquist Rate



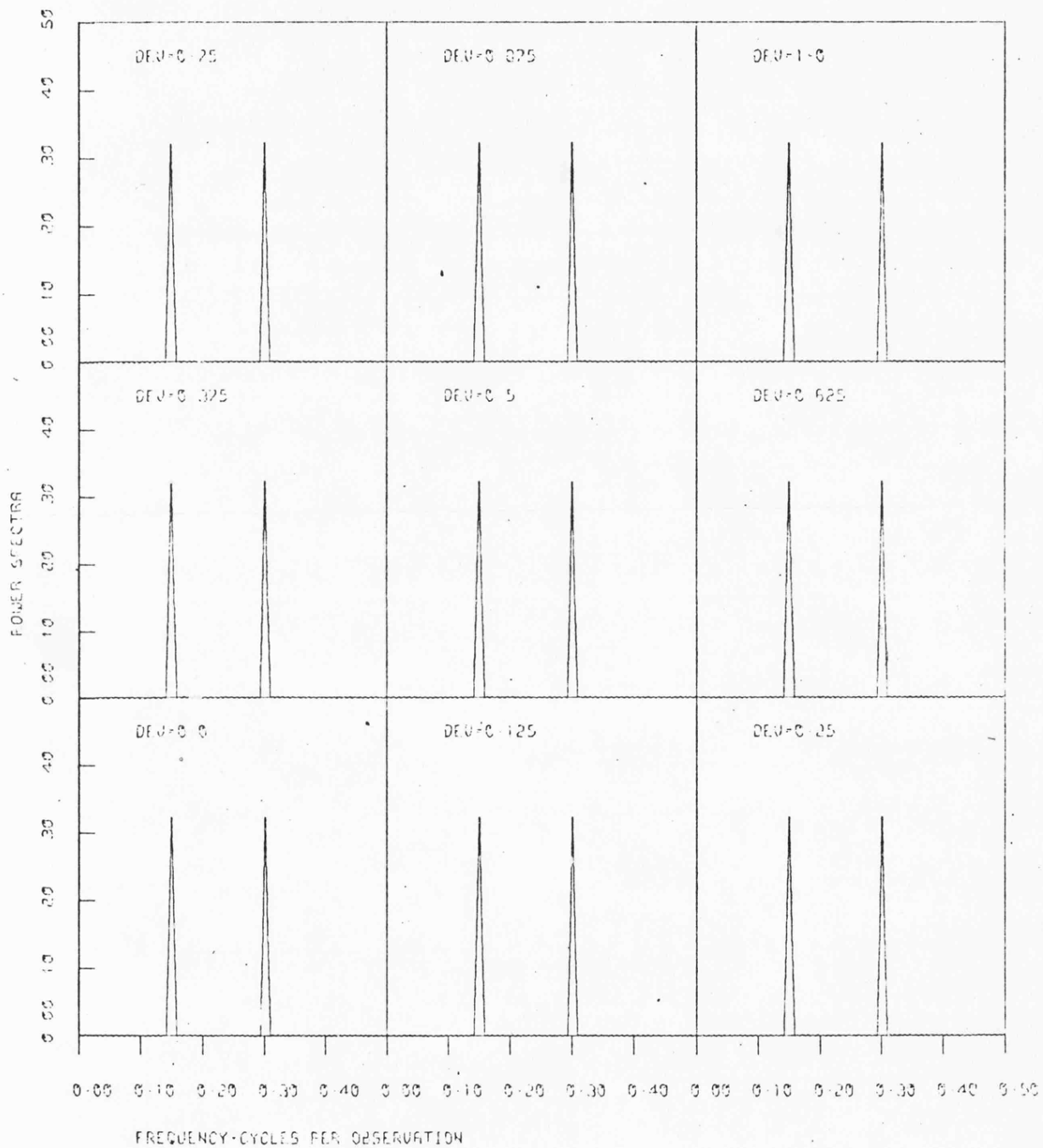
SPECTRAL ANALYSIS

Figure B17. 800 Data Points Sampled at 10.0 times Minimum Nyquist Rate



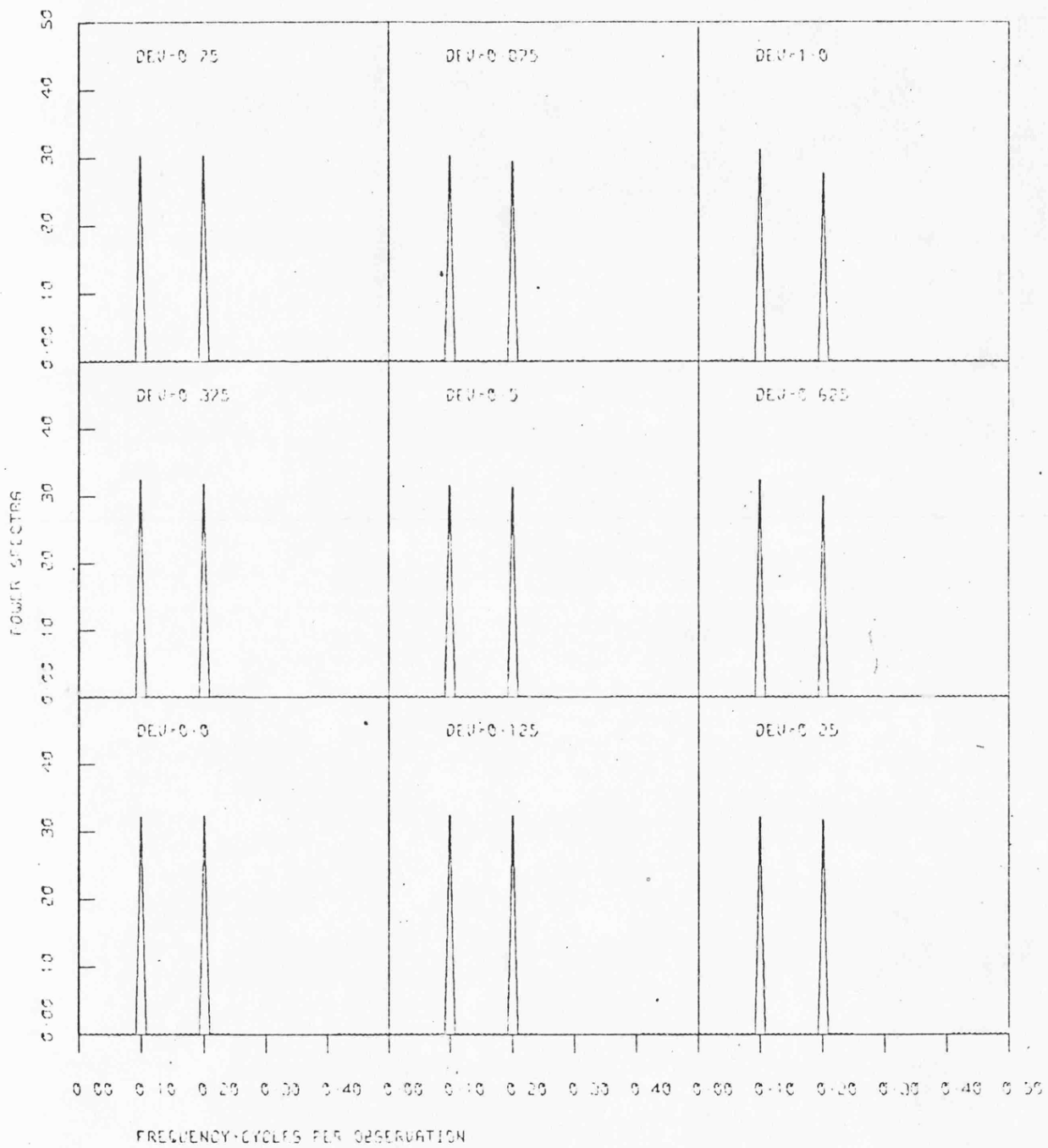
SPECTRAL ANALYSIS

Figure B18. 800 Data Points Sampled at 2.5 times Minimum Nyquist Rate for ω_0 .



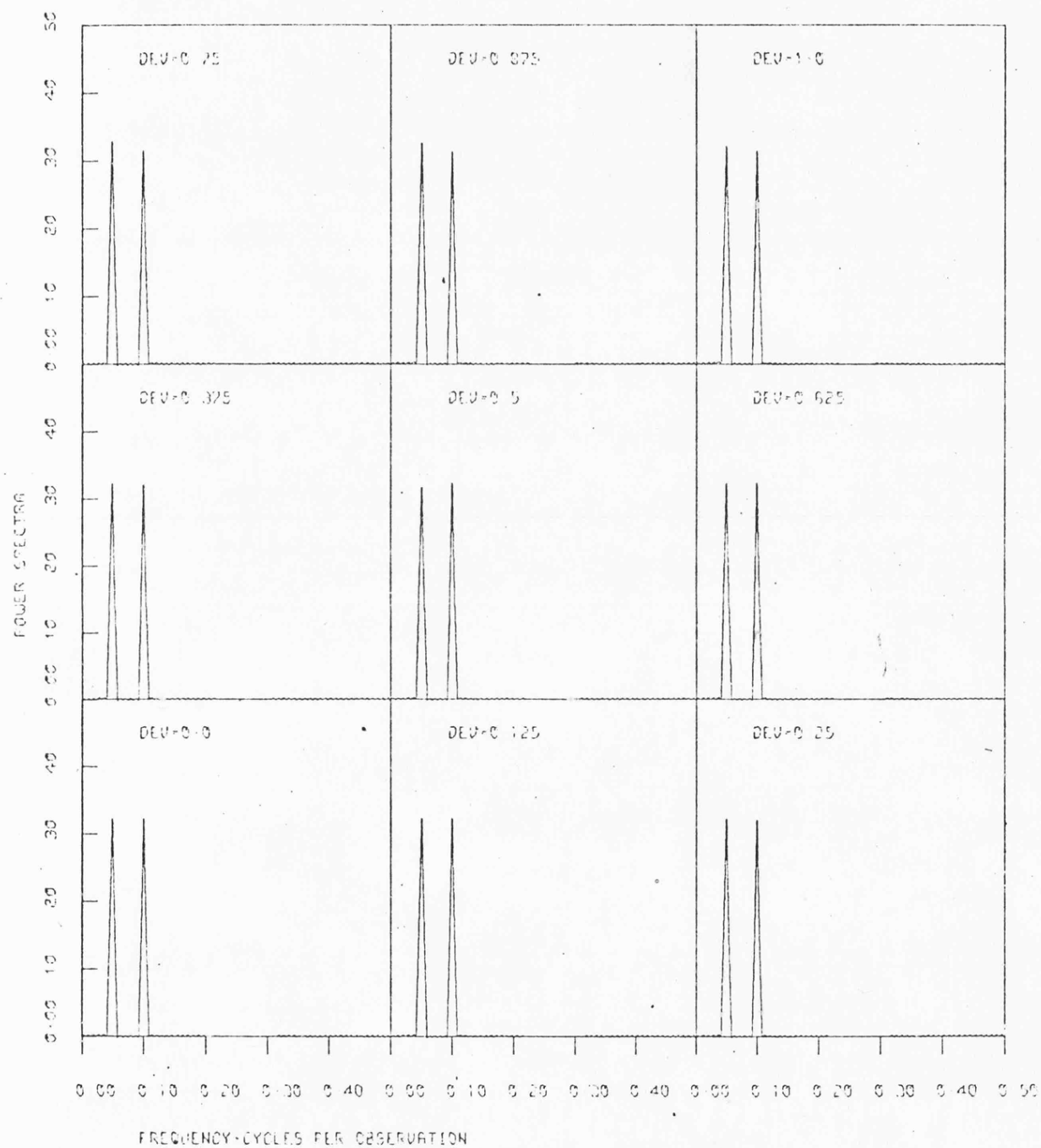
SPECTRAL ANALYSIS

Figure B19. 800 Data Points Sampled at 3.33 times Minimum Nyquist Rate for ω_0



SPECTRAL ANALYSIS

Figure B20. 800 Data Points Sampled at 5.0 times Minimum Nyquist Rate for ω_0



SPECTRAL ANALYSIS

Figure B21. 800 Data Points Sampled at 10.0 times Minimum Nyquist Rate for ω_0

APPENDIX C.

Flow Charts for FOCAL Modifications.

8K Focal Extensions Field 0

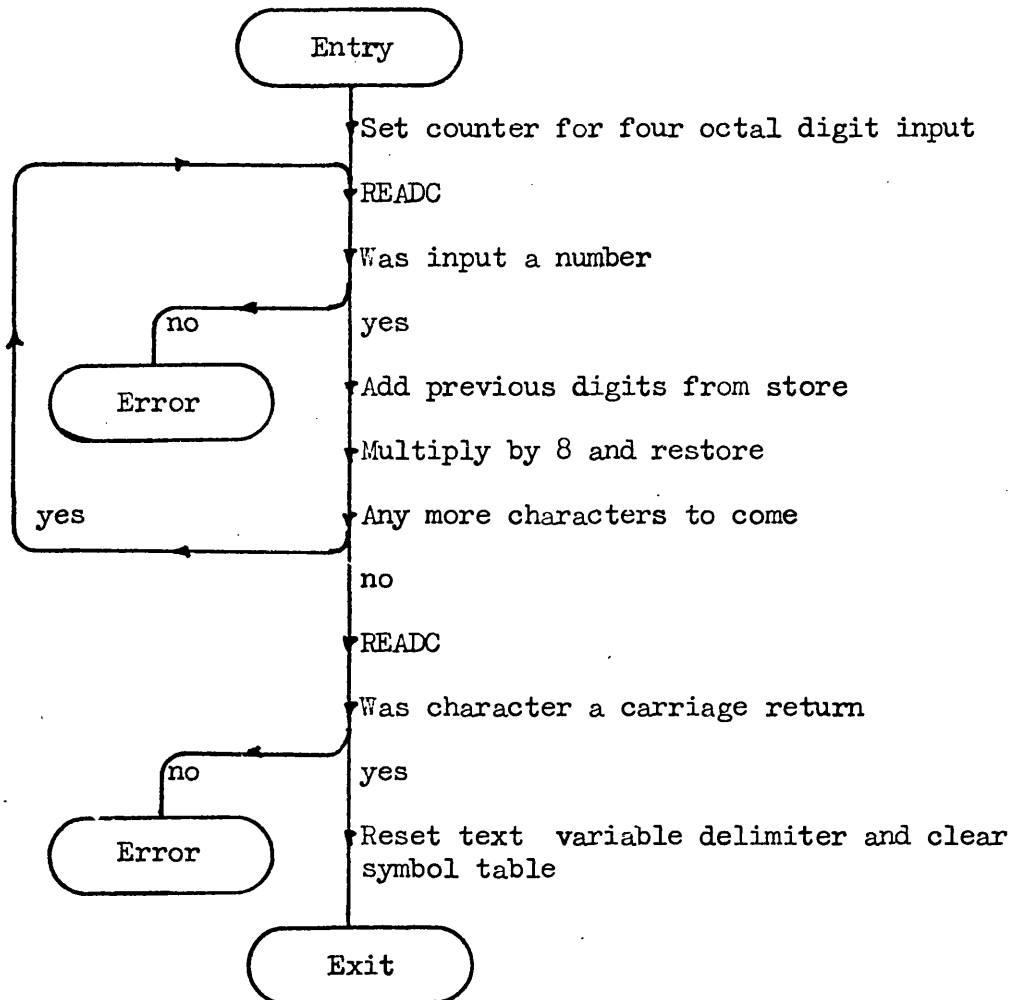
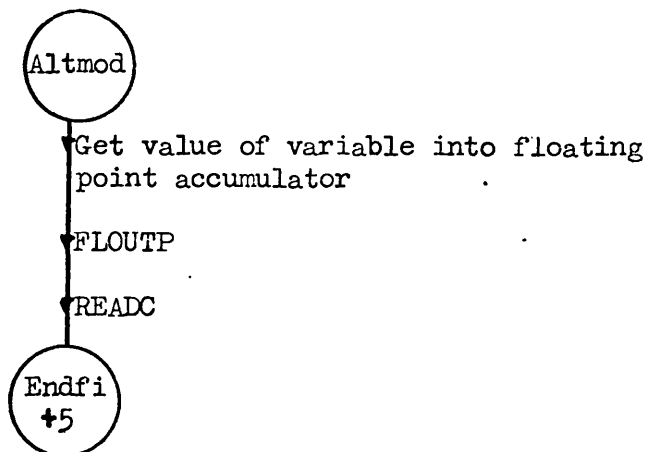
Zero Page Pointers												
Command Decoder					Getln							
Entabf		Do		Pushdown List Controls								
Control & Transfer		Write		Testo		Sortc		Grptst		Input		
Comlst		If		Set & For		Dys & Int		Comgo				
Type & Ask		Modify		Sortj		Adc,Outl and others						
Getarg					Spnor,Testn,Ran,Popj and others							
Ecall												
Terms Sgn,Abs et al		Tstlpr,Partest		Delete		Readc		Entabl				
Erase		Findln		Geto		Endln		Extensions				
Hsp,Prntln,Prnt and Printc		Packc										
Interrupt Processor , I/O Routines and Error Recovery Routine												
Rubout		Symbol Table Dump		O/P buffer		Psymb8,Gsym8,Gsub8						
Extensions to Library and Variable search												
Echo disable and Power Failure Routines												
Fran												

8K Focal Extensions Field1

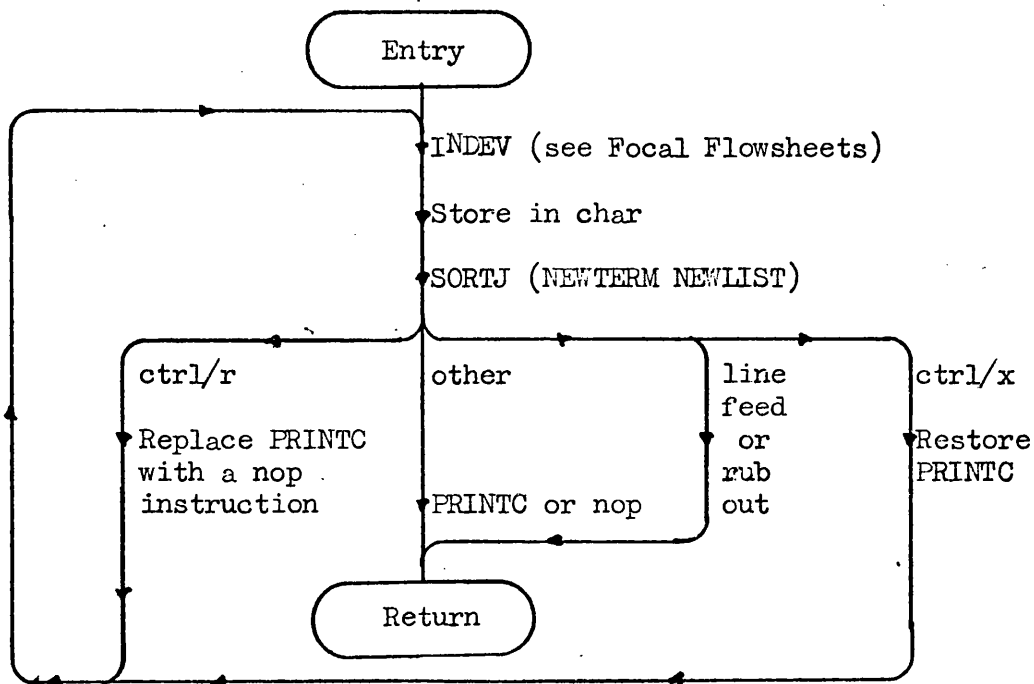
Command Input Buffer	
0	
1	
2	
3	
4	
5	
6	
7	
10	TEXT
11	
12	
13	
14	
15	
16	
17	
20	
21	
22	
23	
24	
25	
26	
27	VARIABLES
30	
31	
32	
33	
34	
35	
36	← PUSHDOWN ITST
37	Loaders

BREAK . Text/variable trade-off routine

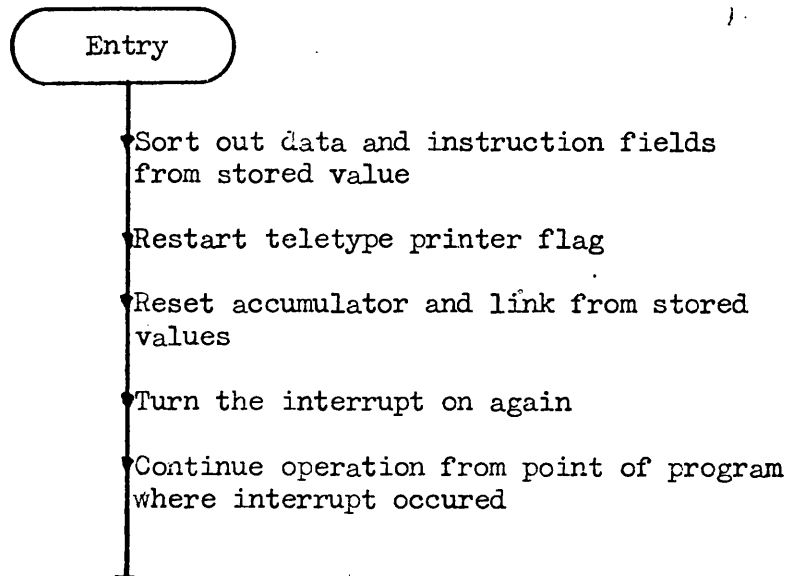
Entered as an extension to the library command

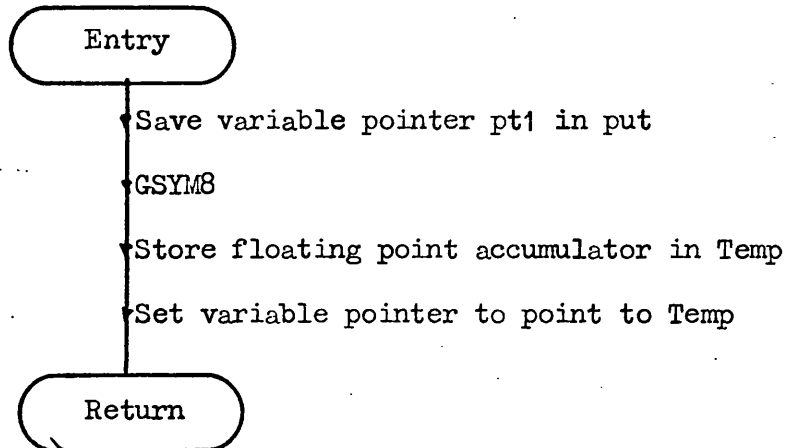
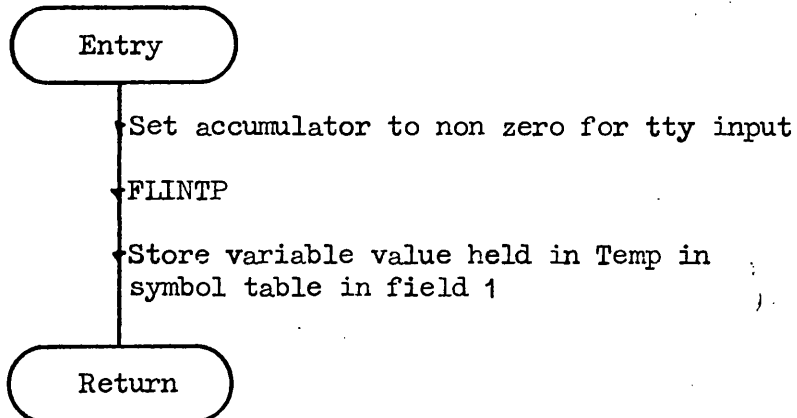
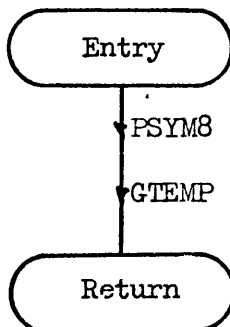
Extension to Altmode reply to Ask Command

Modification to Input Routine CHIN for input echo suppression



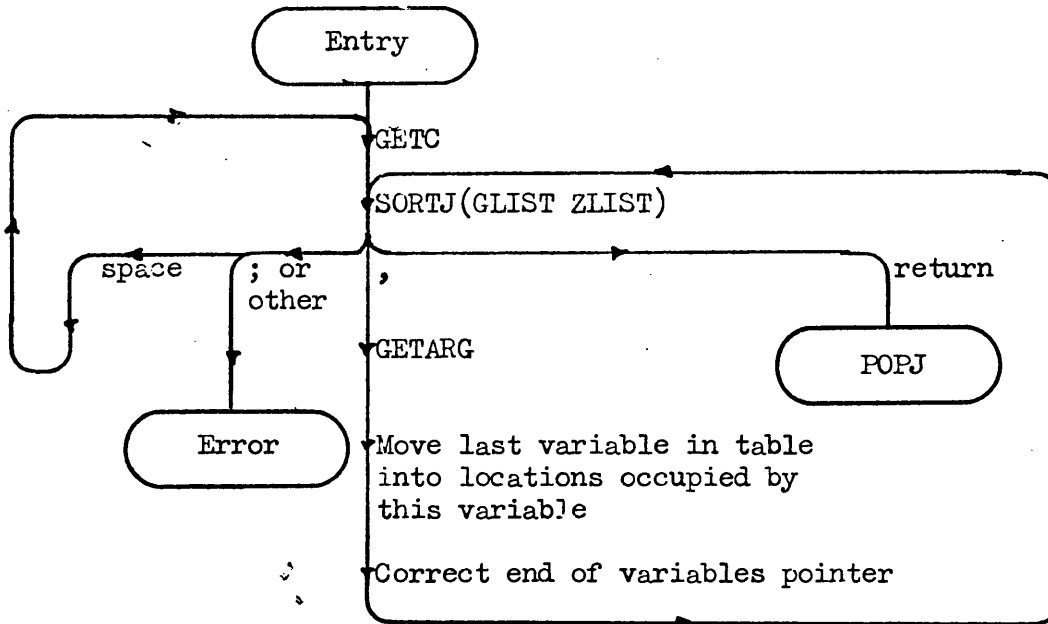
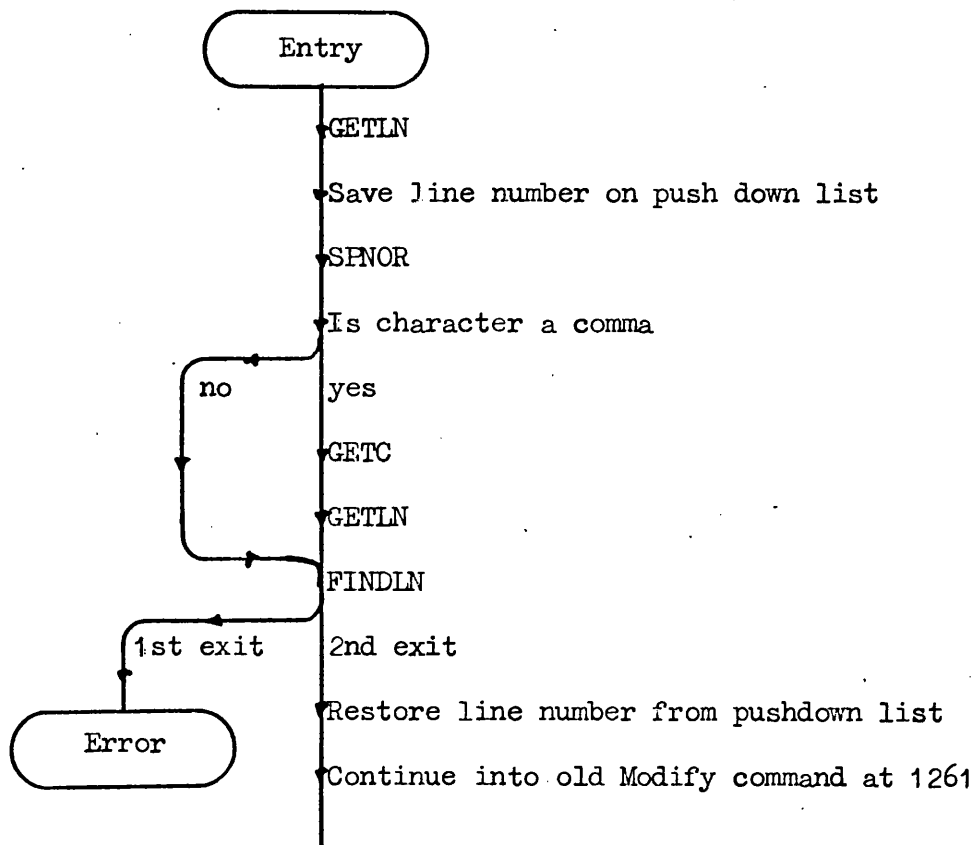
Power on Automatic Restart Routine

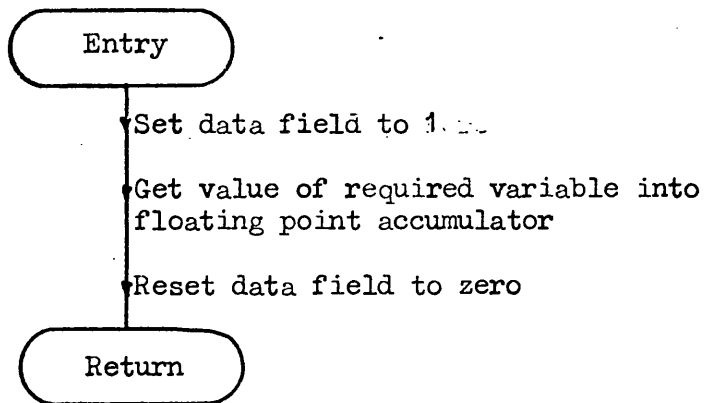
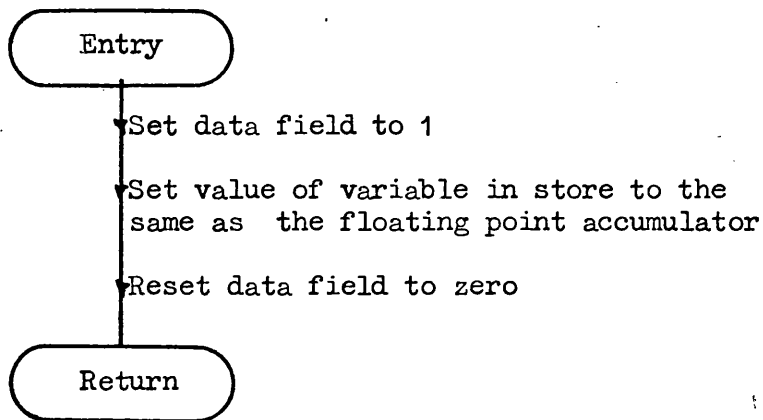
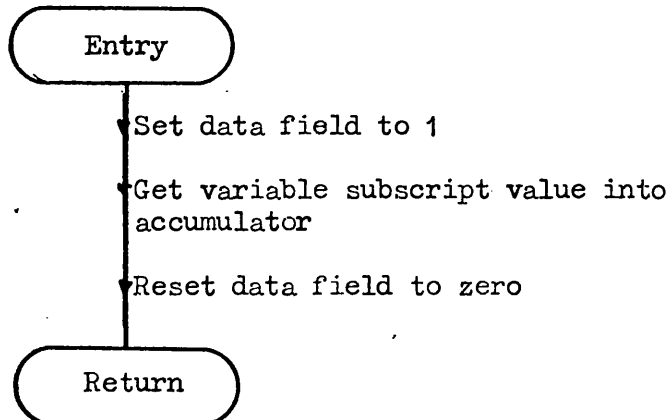


GTEMPPUTSYMPSYMP

Variable Erase routine

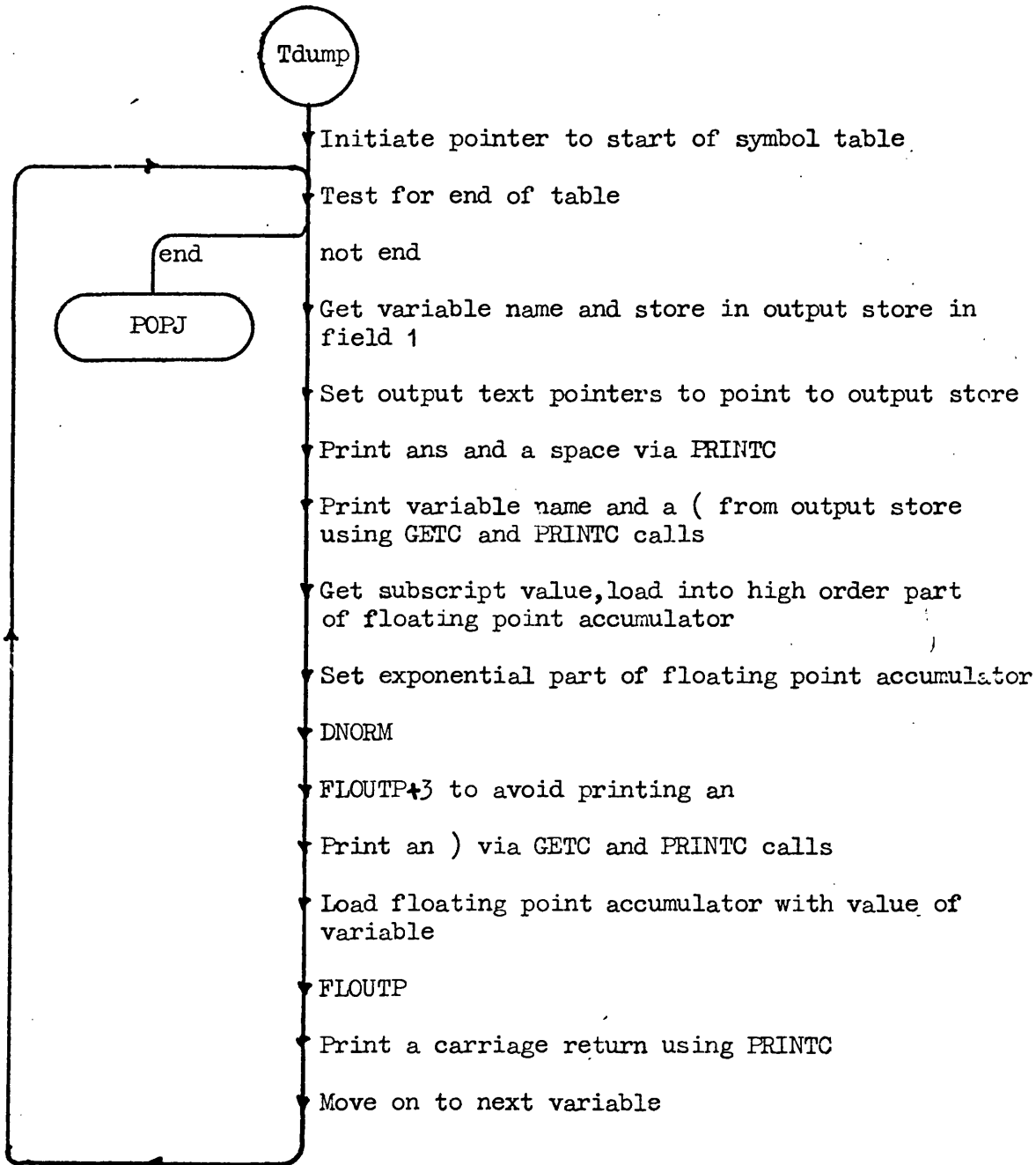
Entered from function error call in Erase command

Extension to Modify Command for line duplication

GSYM8PSYM8GSUB8

Extension to Symbol Table Dump Routine

Prints an S for SET before the variable name and value so that symbol table can be saved on paper tape and read again

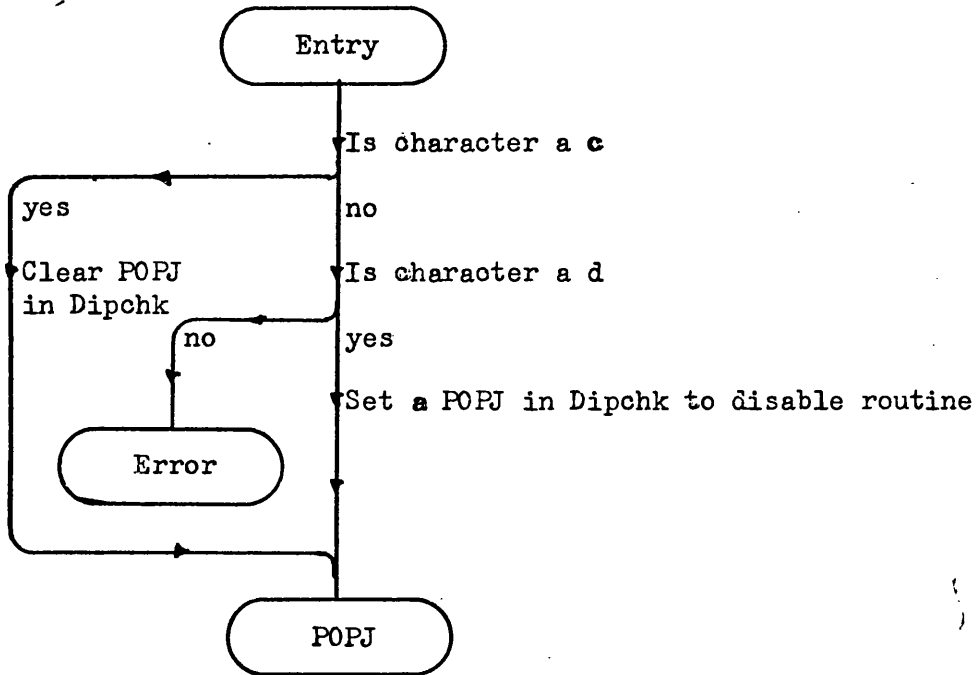


DIPCHK Enable and Disable Commands

E C enables the routine

E D disables the routine

Entered from an Error call in Erase Command



APPENDIX D.

Flow Charts for Single User

Real-Time FOCAL

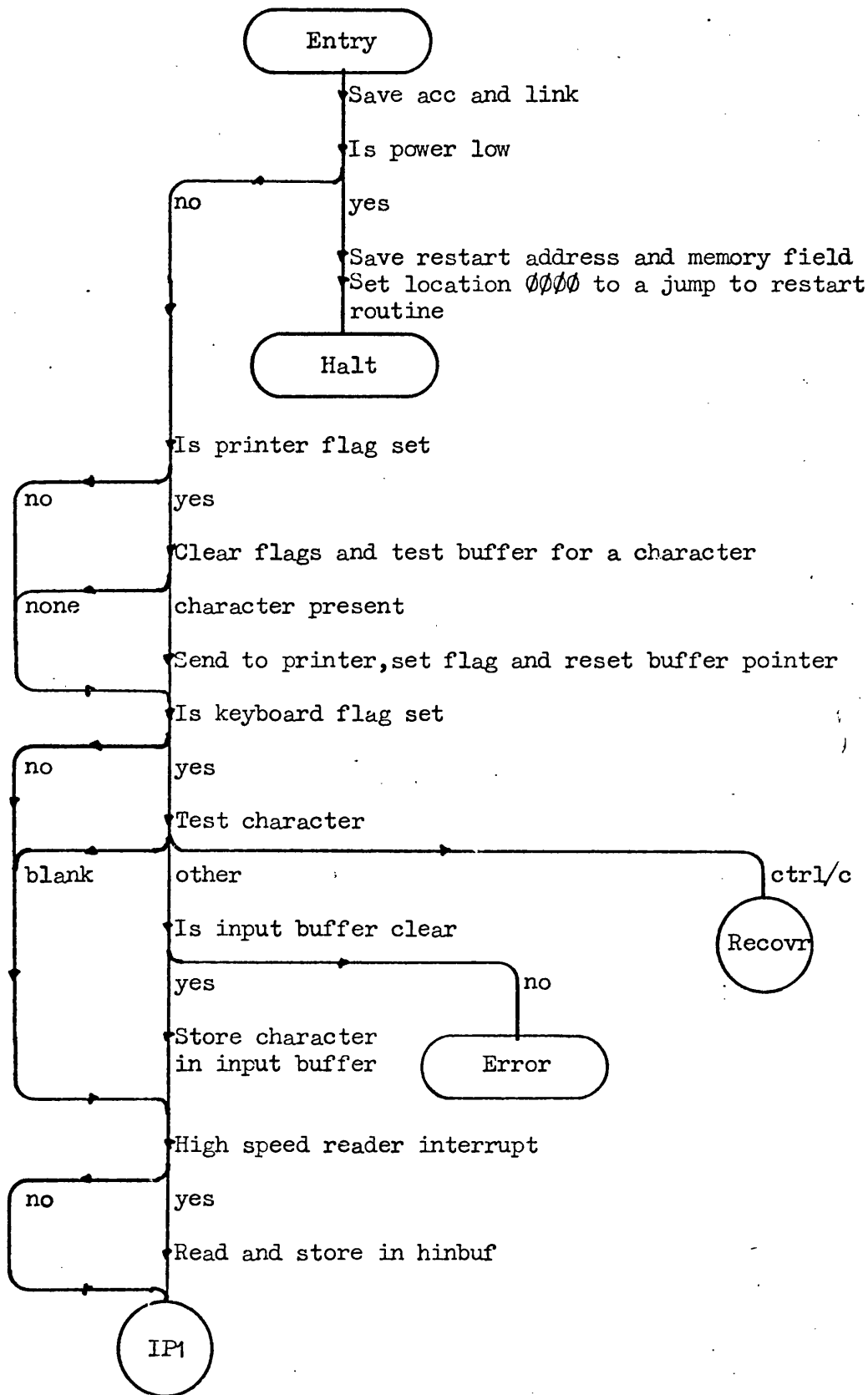
Real Time Focal : Dym Core map Field 0

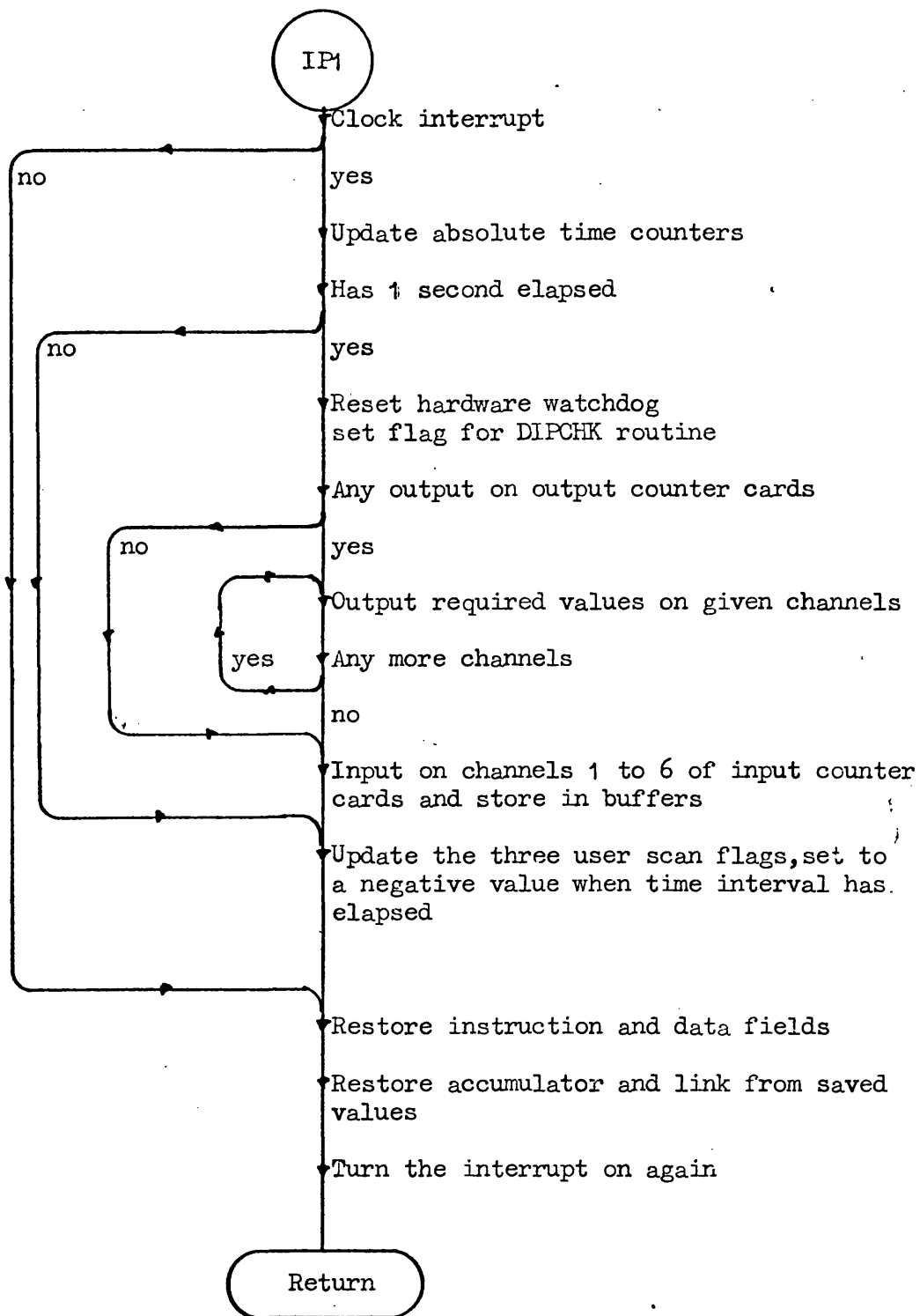
Zero Page Pointers											
Command Decoder				Getln							
Fntabf		Do		Pushdown List Controls							
Control & Transfer		Write		Testc		Sortc		Grptst		Input	
Comlst		If		Set & For				Dys & Int		Congo	
Type & Ask				Modify		Sort,j		Adc,Outl and others			
Getarg				Spmar,Testn,Ran,Popj and others							
Ecall											
Terms Sgn,Abs et al		Tstlpr,Partest		Delete		Readc		Fntabl			
Erase		Findln		Getc		Endln		Extensions			
Hsp,Prntln,Prnt and Printc				Packc							
Interrupt Processor , I/O Routines and Error Recovery Routine											
Rubout		Symbol Table Dump		C/P buffer		Gsym8,Pyym8,Gsub8					
Extensions to Library & Variable Search				Echo Disable		Dipchk,Sortnset, Setadd					
Parameter Modification Routine											
Clock Service Routine Extension to the Interrupt Processor						Flag continued					
Getarg(*)		Power Failure Routines		Flag		Flim		Lists			
Fin and Fout Commands											
PCI Control Algorithm				Finc		PCI Algorithm Data Table					
Synchronous Service Routines		Frn		Feedforward Control Compensation Algorithm Dym							
Dyn Data Table		Cos and Sin				Mod and Altnode					
FLOATING POINT											
INPUT AND OUTPUT											
ROUTINES											
						High Speed Reader Routine					
FLOATING POINT INTERPRETER											
Variable Phrase Routine		E C, E D				Library		Further Extensions & Hsp Switch			

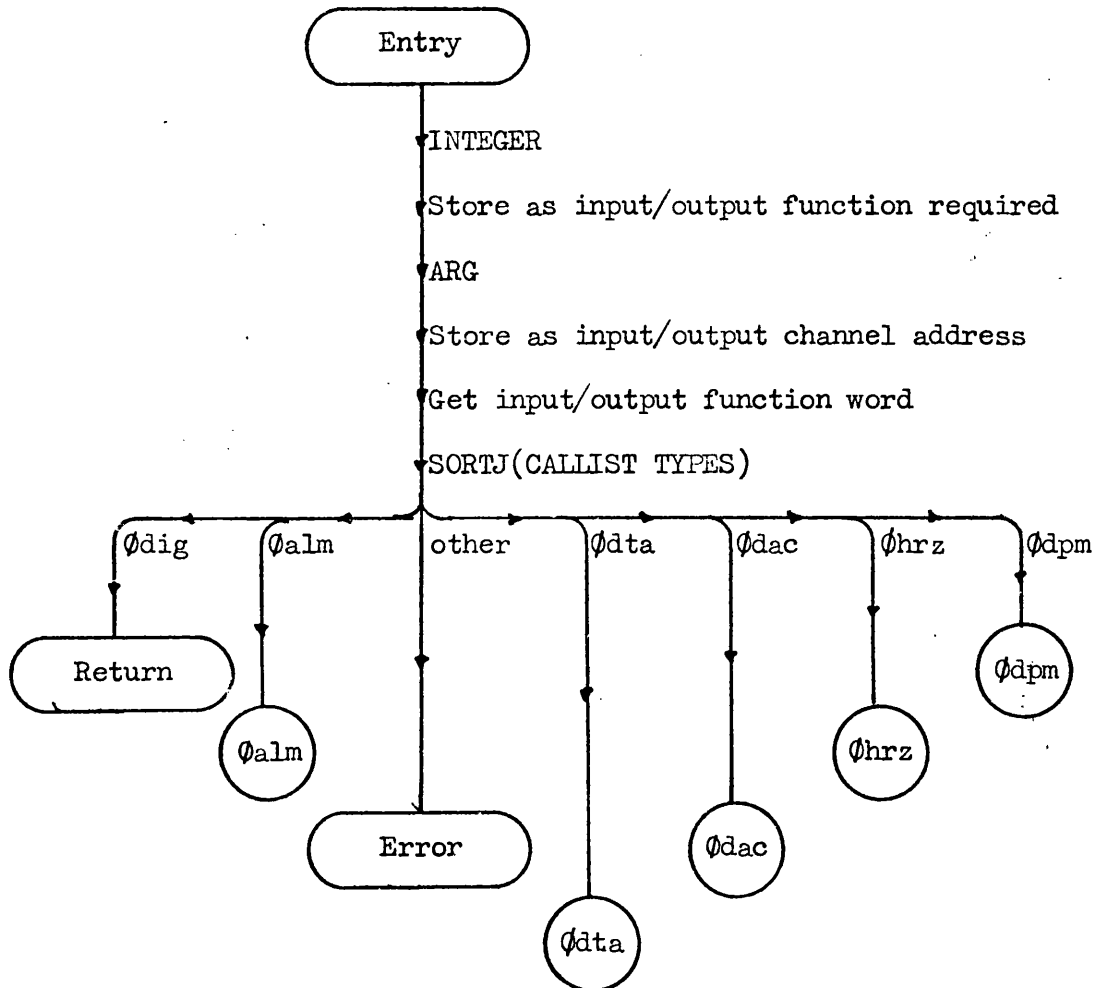
0	Real Time Focal : Dyn Core map Field 1
1	Command Input Buffer
2	
3	
4	
5	
6	
7	
10	
11	TEXT
12	
13	
14	
15	
16	
17	
20	
21	
22	
23	
24	
25	
26	
27	VARIABLES
30	
31	
32	
33	
34	
35	
36	RUSHDOWN LIST
37	Loaders

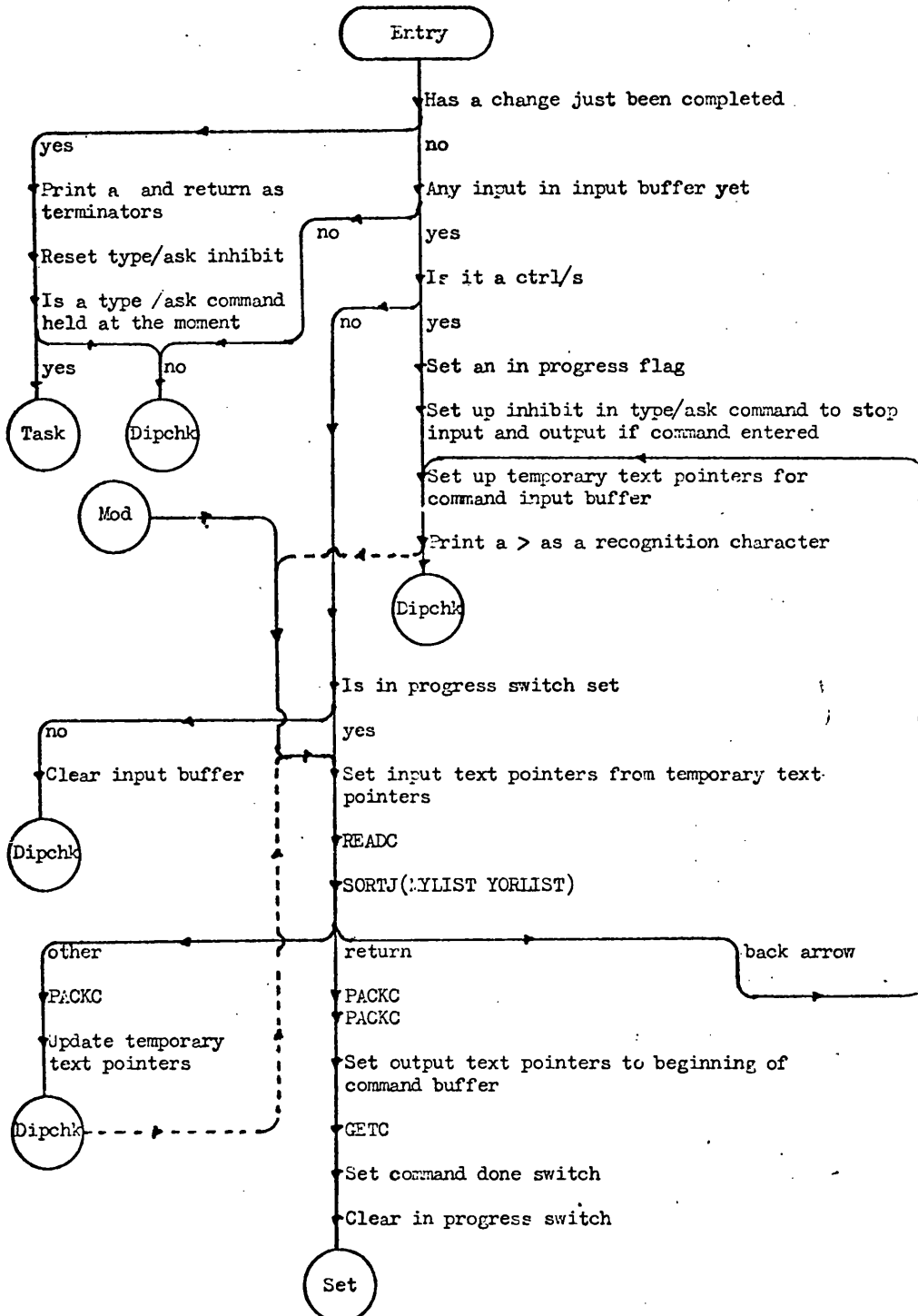
Real Time Focal : Log Core map Field 1

0	Command Input Buffer
1	
2	
3	
4	
5	
6	
7	
10	TEXT
11	
12	
13	
14	
15	
16	
17	
20	
21	
22	
23	
24	
25	
26	
27	VARIABLES
30	
31	
32	
33	
34	
35	
36	PUSHDOWN LIST
37	Loaders

Extensions to Focal Interrupt Processor



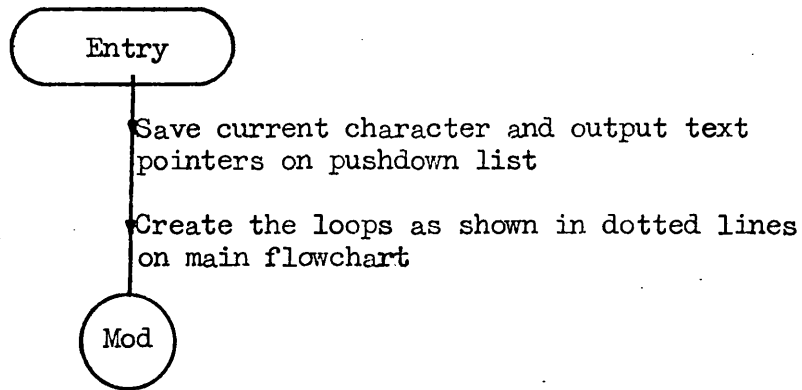
SORTNSET

On-line Variable Modification Routine

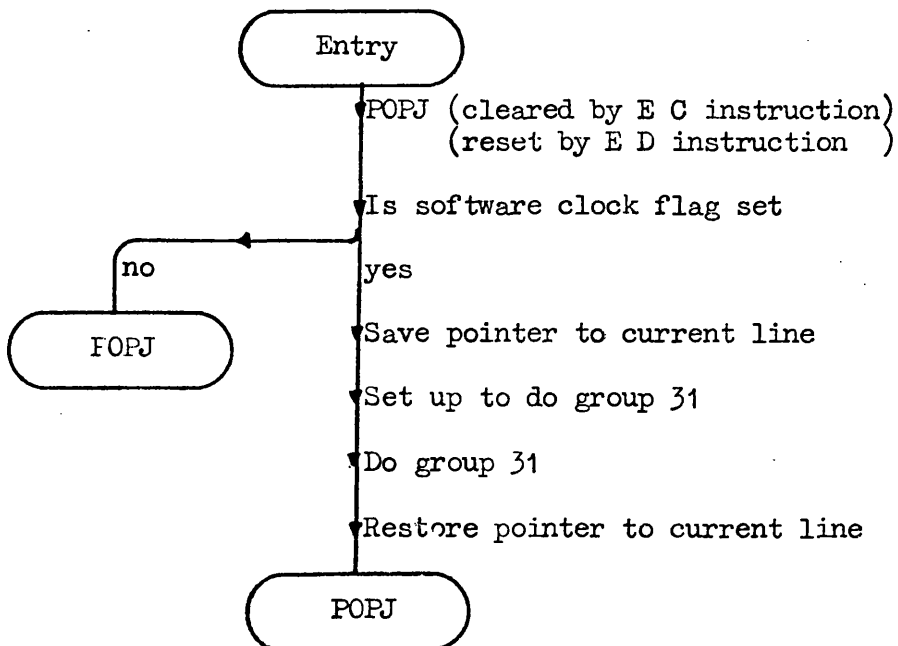
The dotted lines show the internal loops made when type/ask command is put into inhibit mode

Type/ask inhibit routine

Entered when user is in parameter modification mode and the program enters a type or ask command

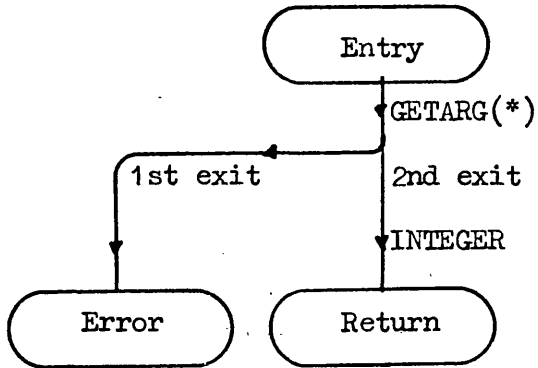
Dipchk

Entered from variable modification routine

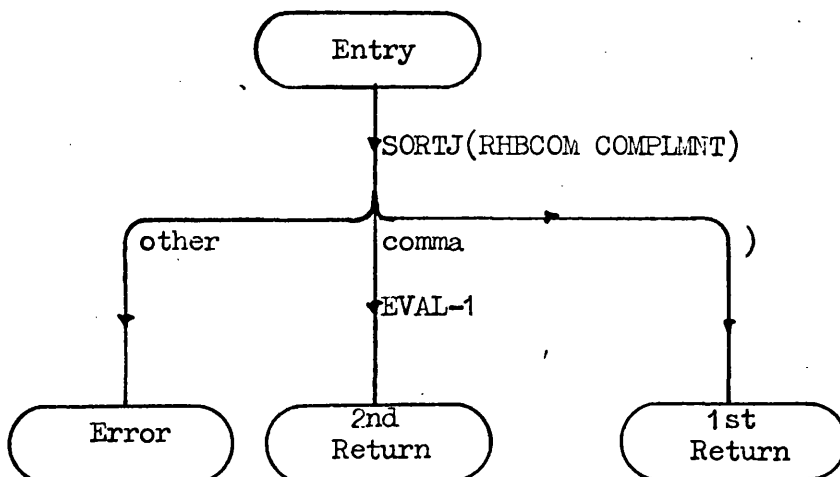


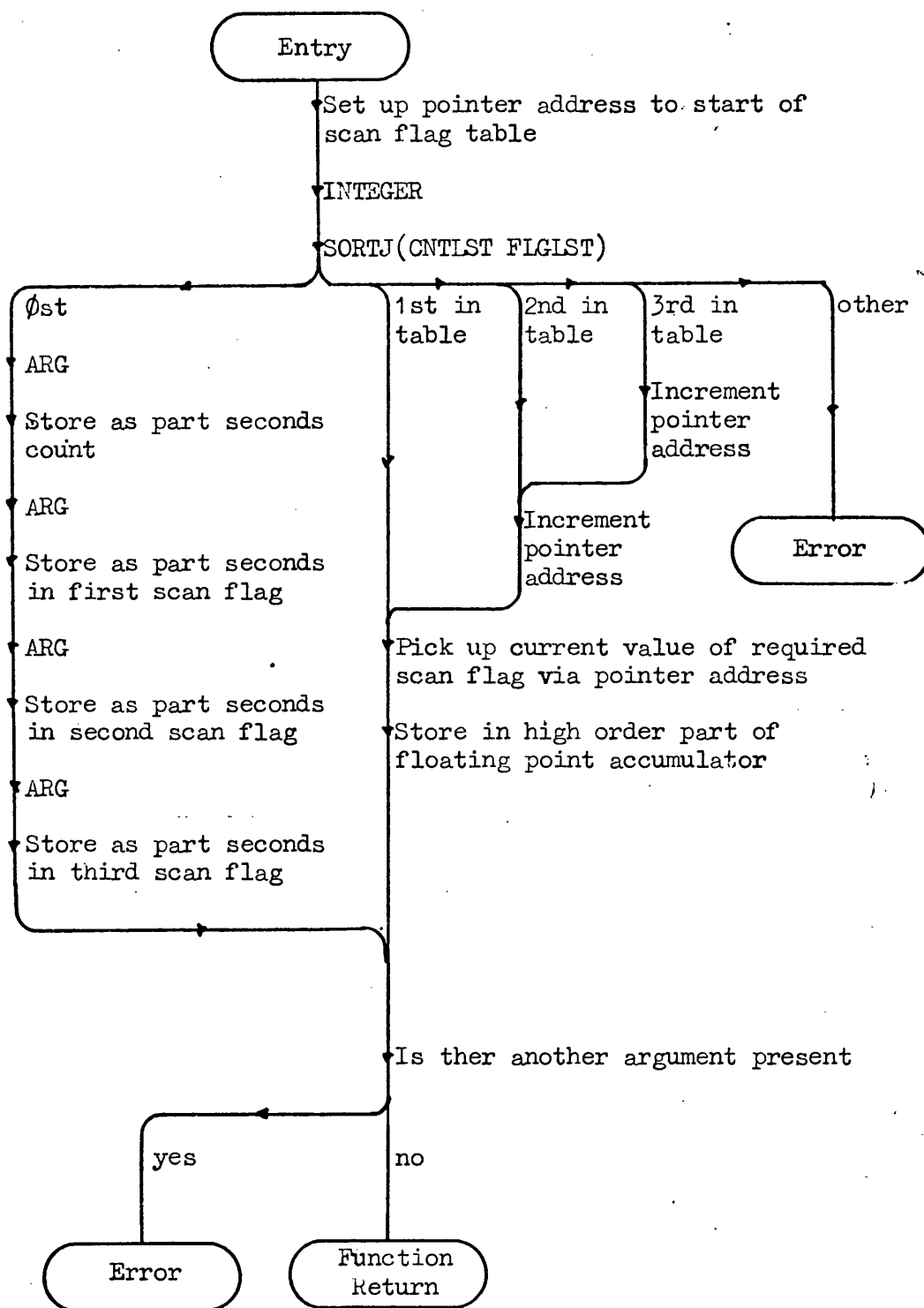
ARG

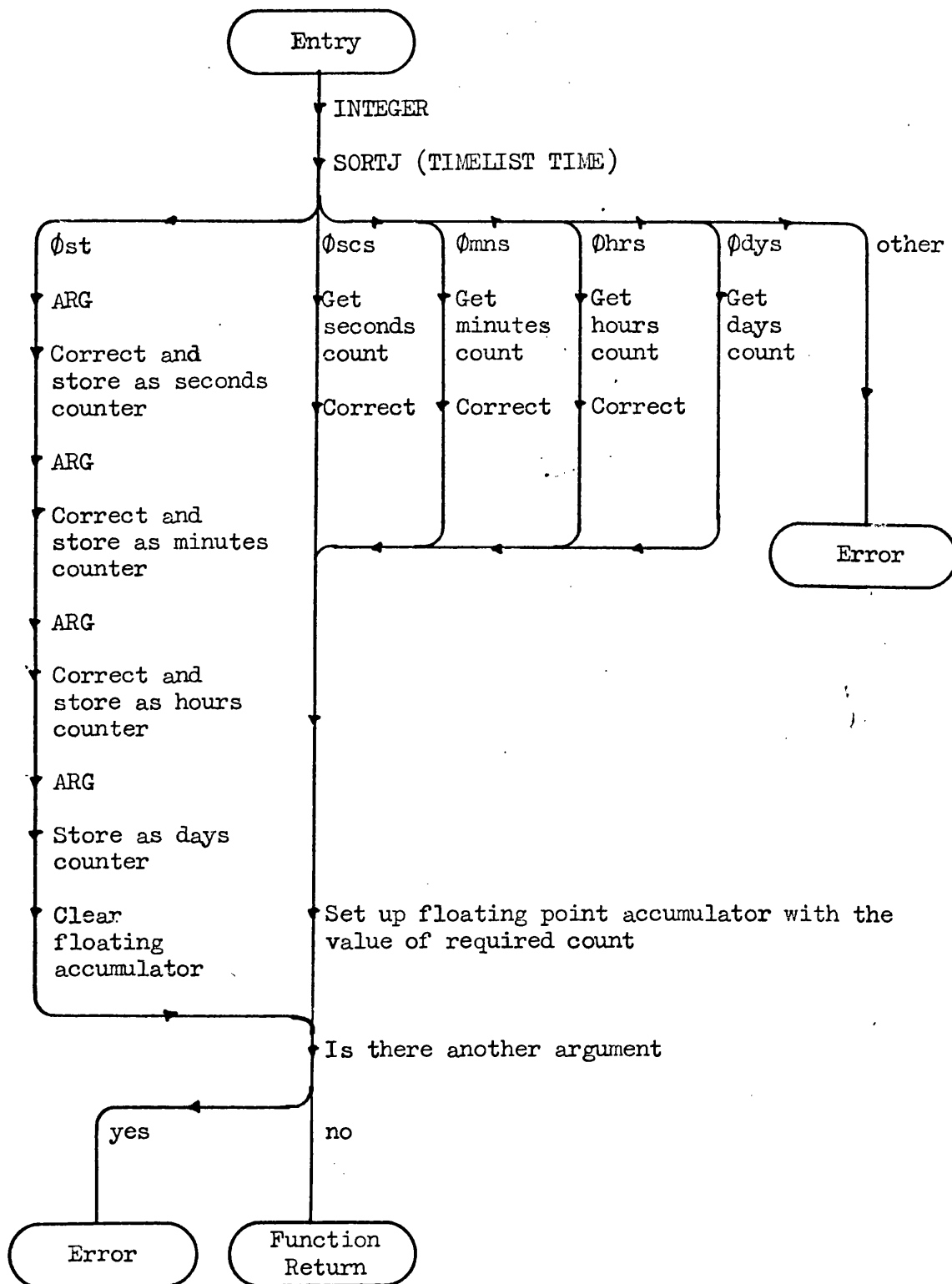
Obtain next argument and convert to integer

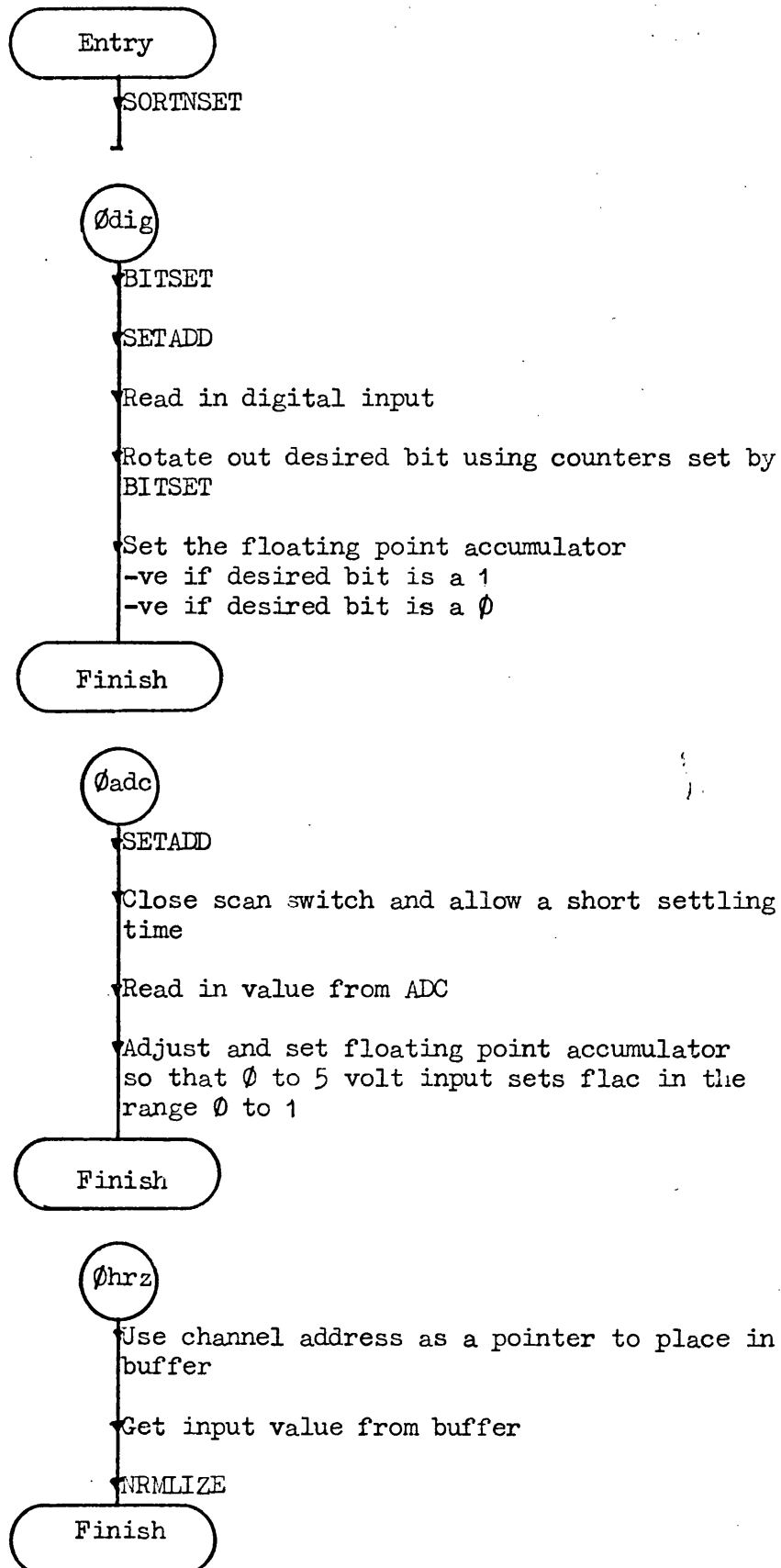
NEXT

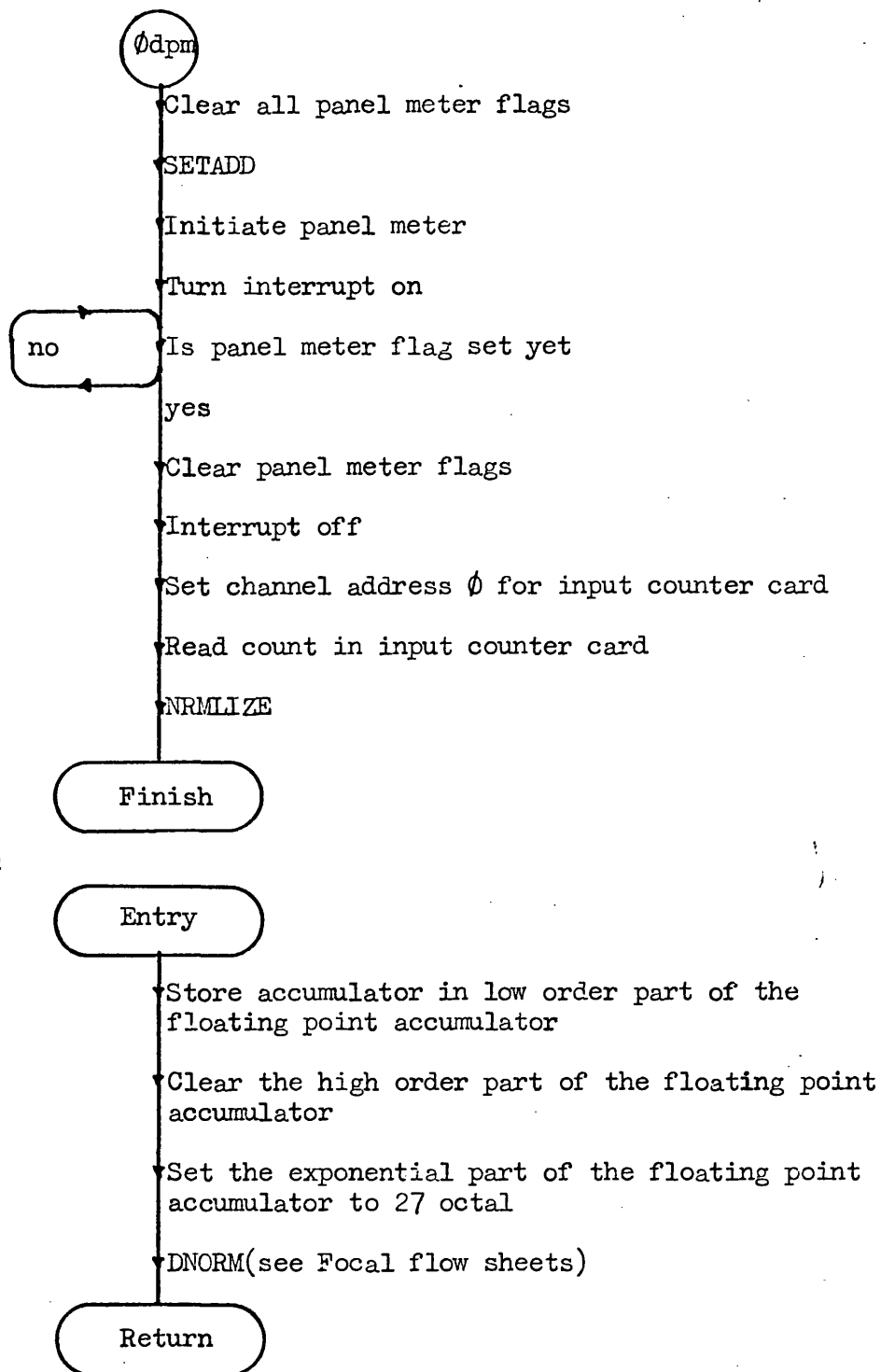
Called by GETARG(*) which should not be confused with
Focals variable search routine GETARG



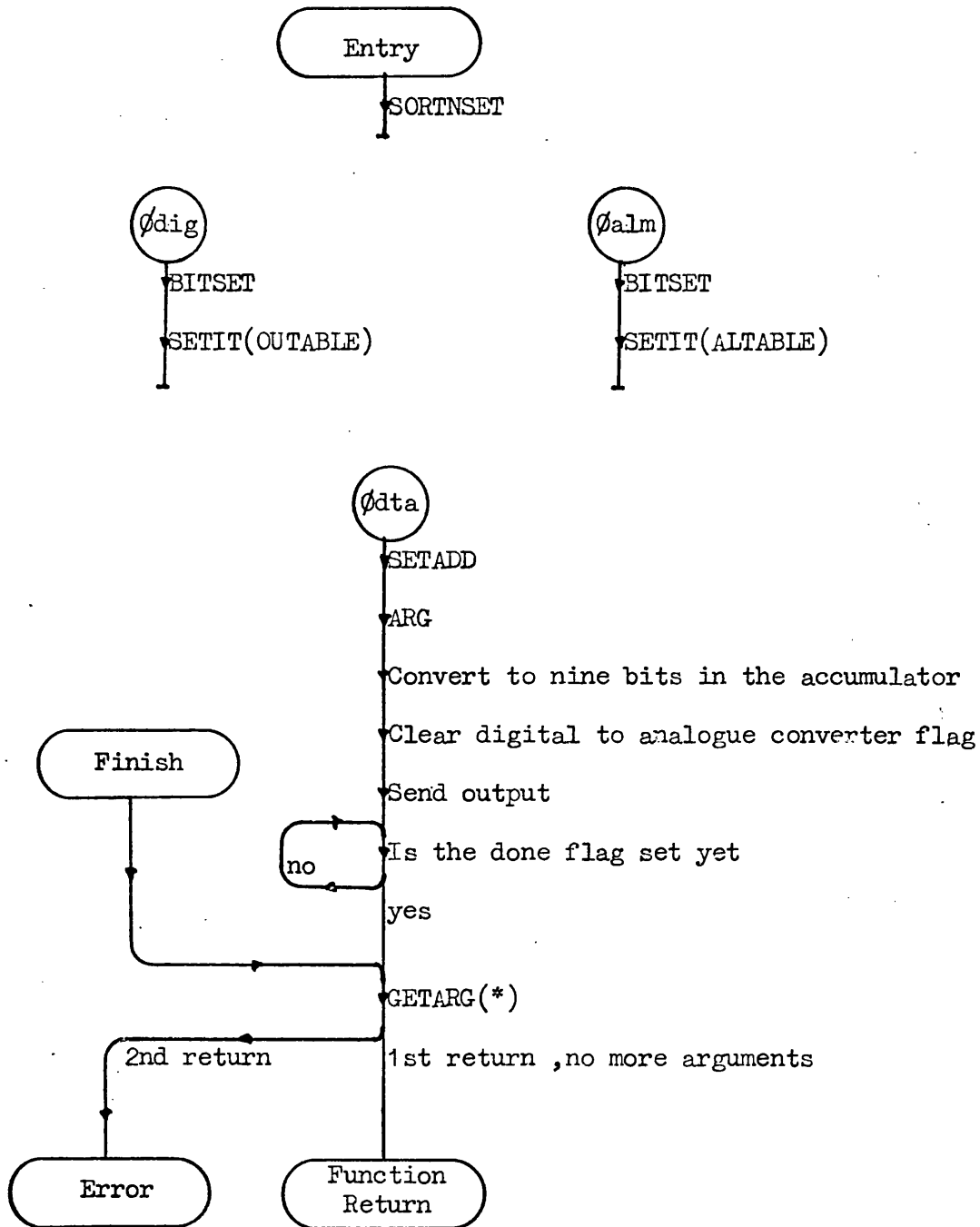
Flag Routine for Real Time Focal

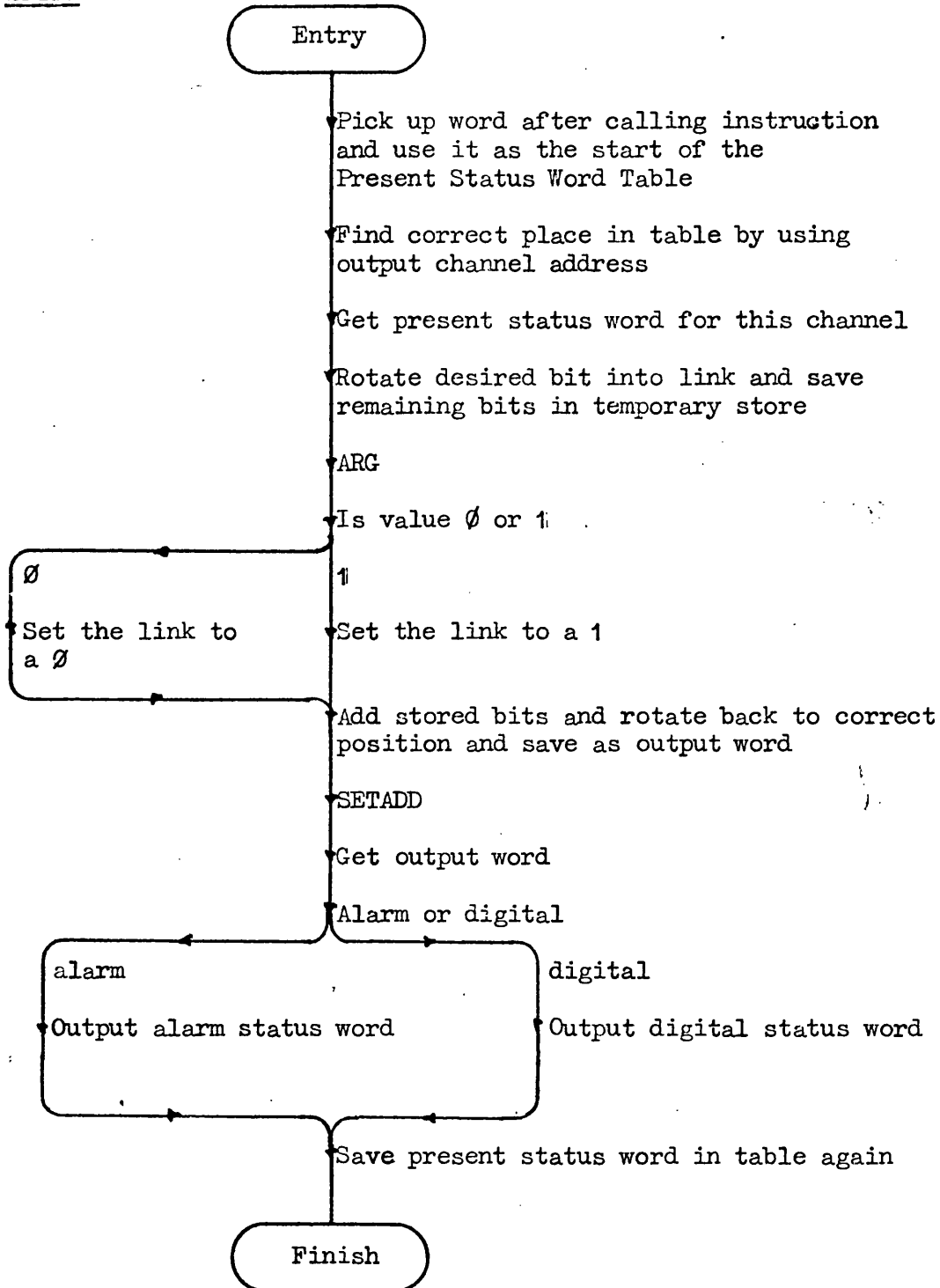
Time Routine for Real Time Focal

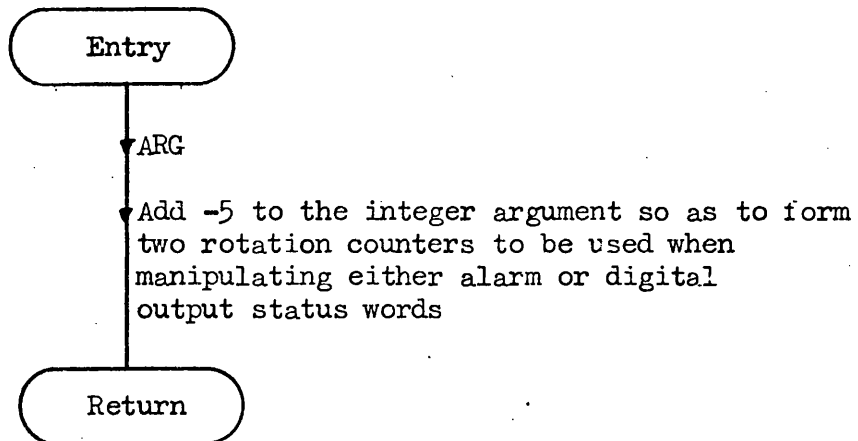
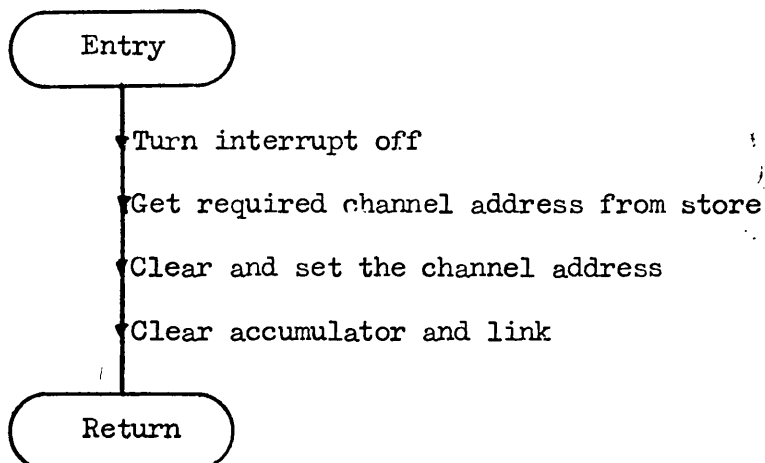
Input Routines called from the Function FIN

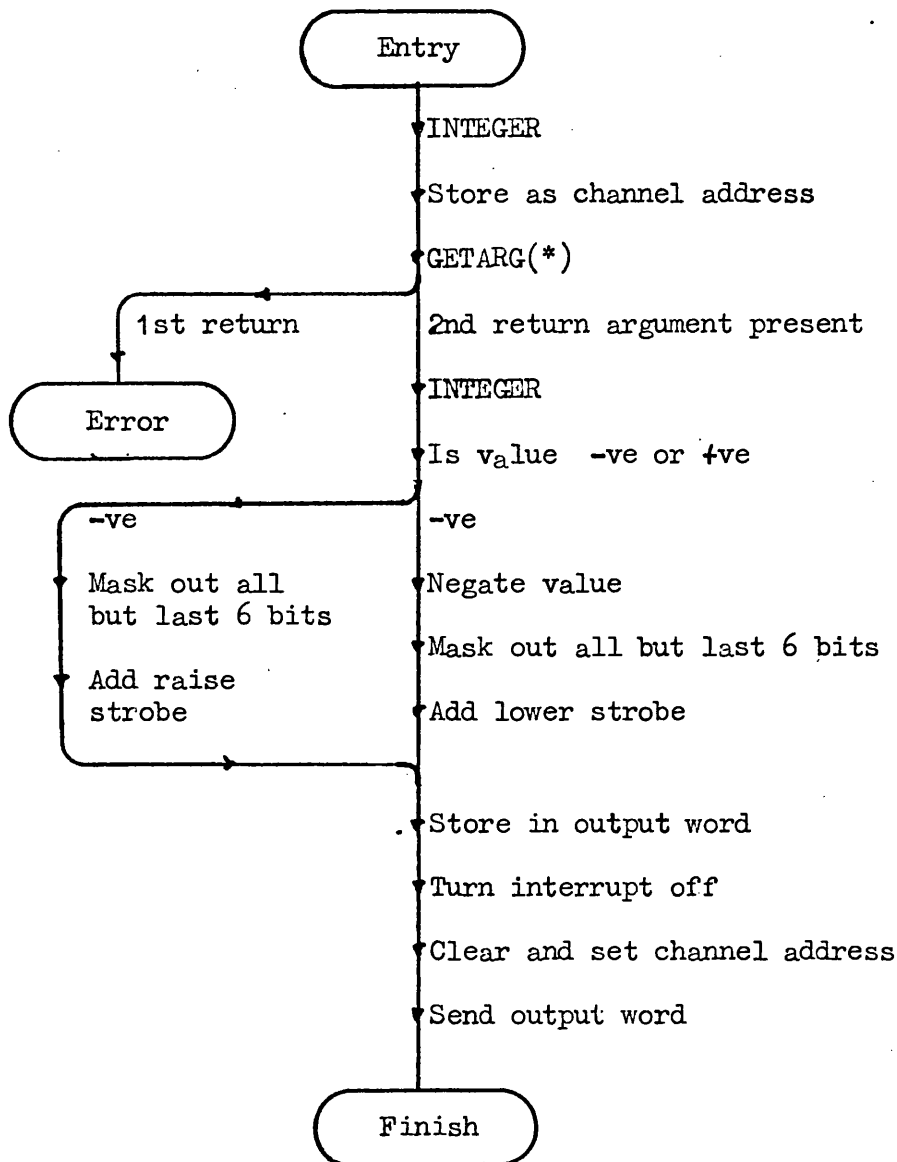
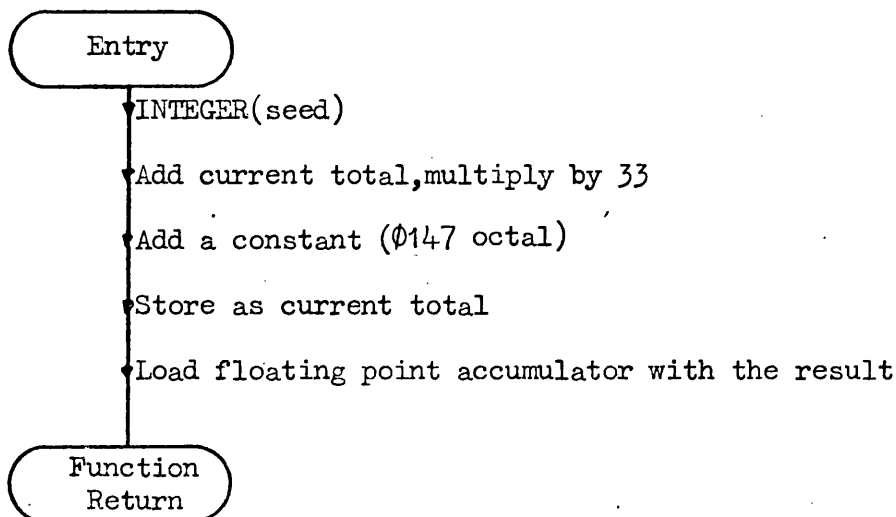


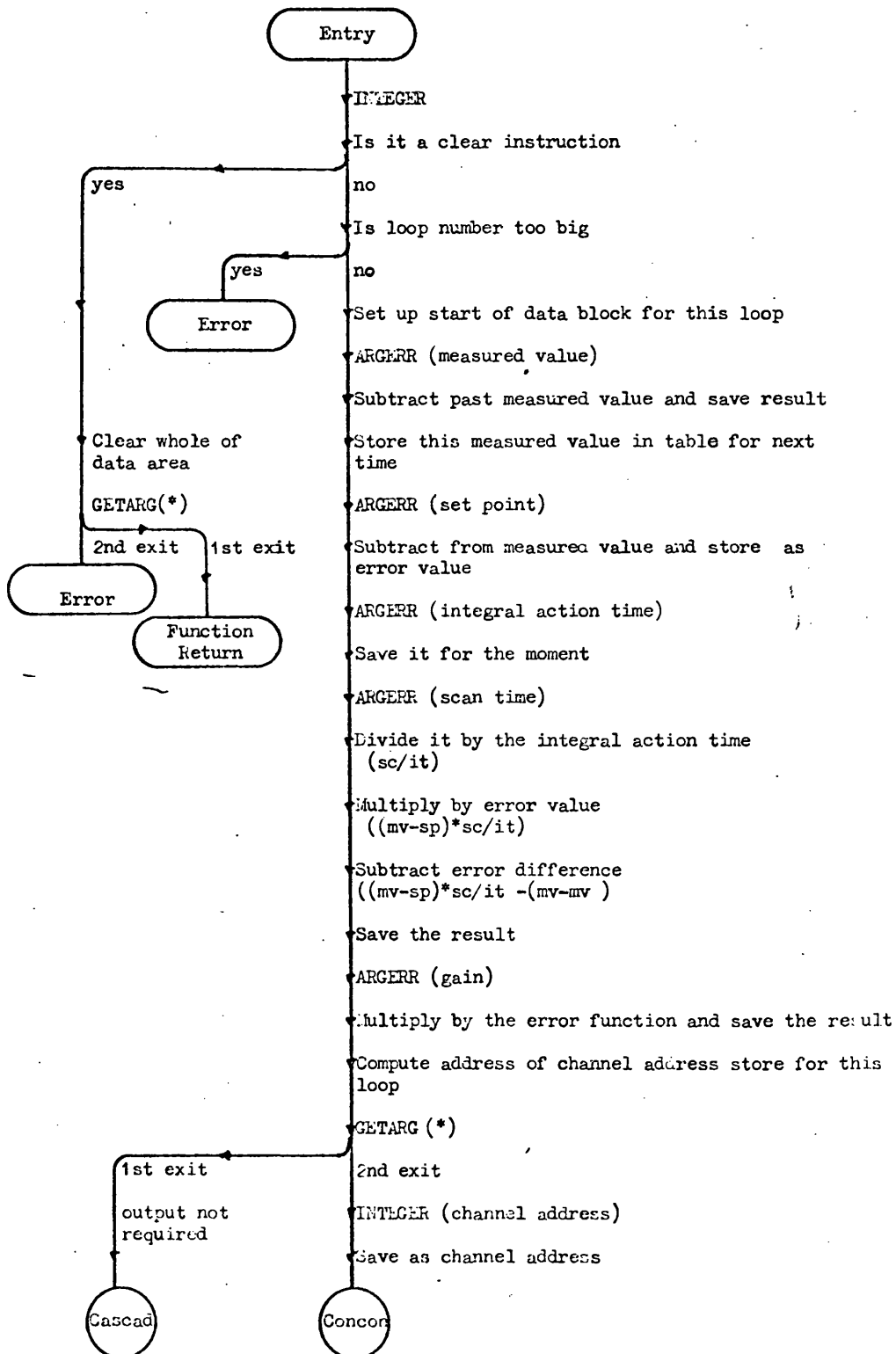
Output Routines called from the Function FOUT

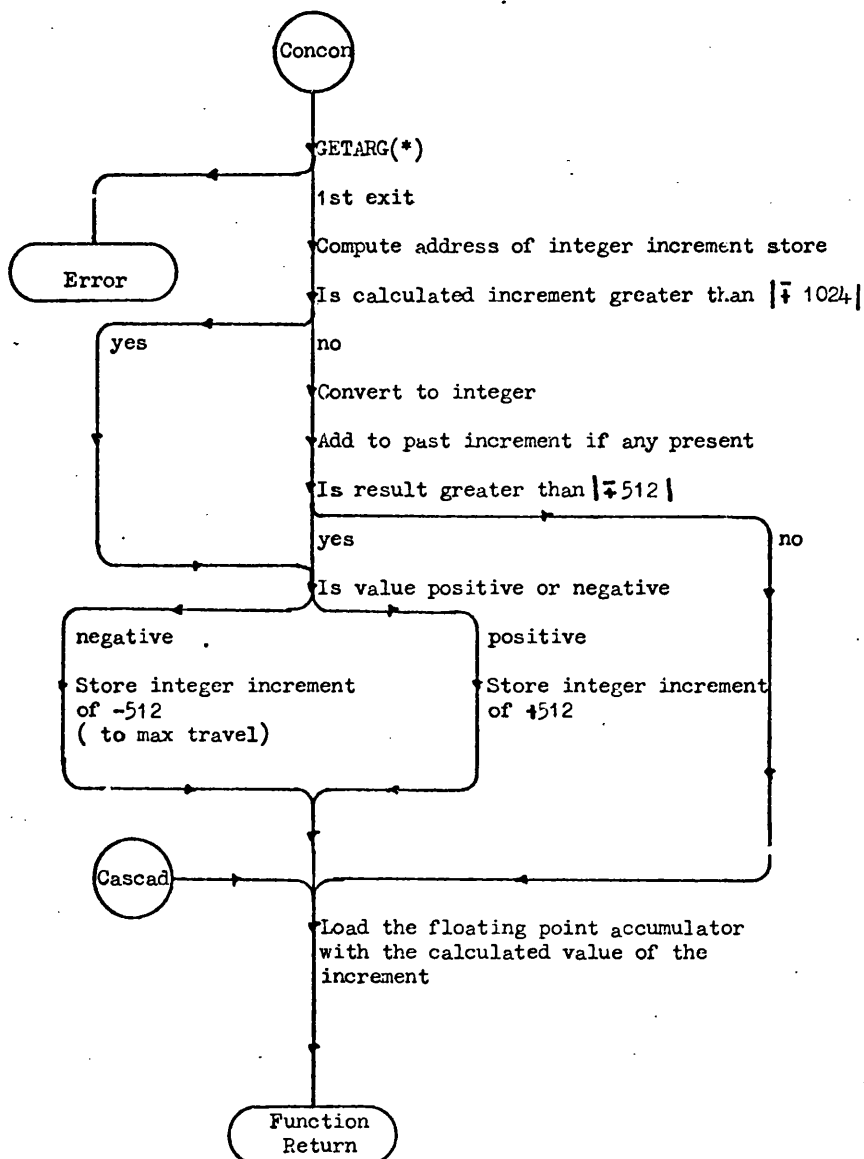
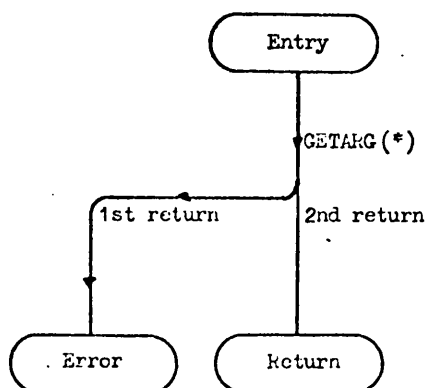


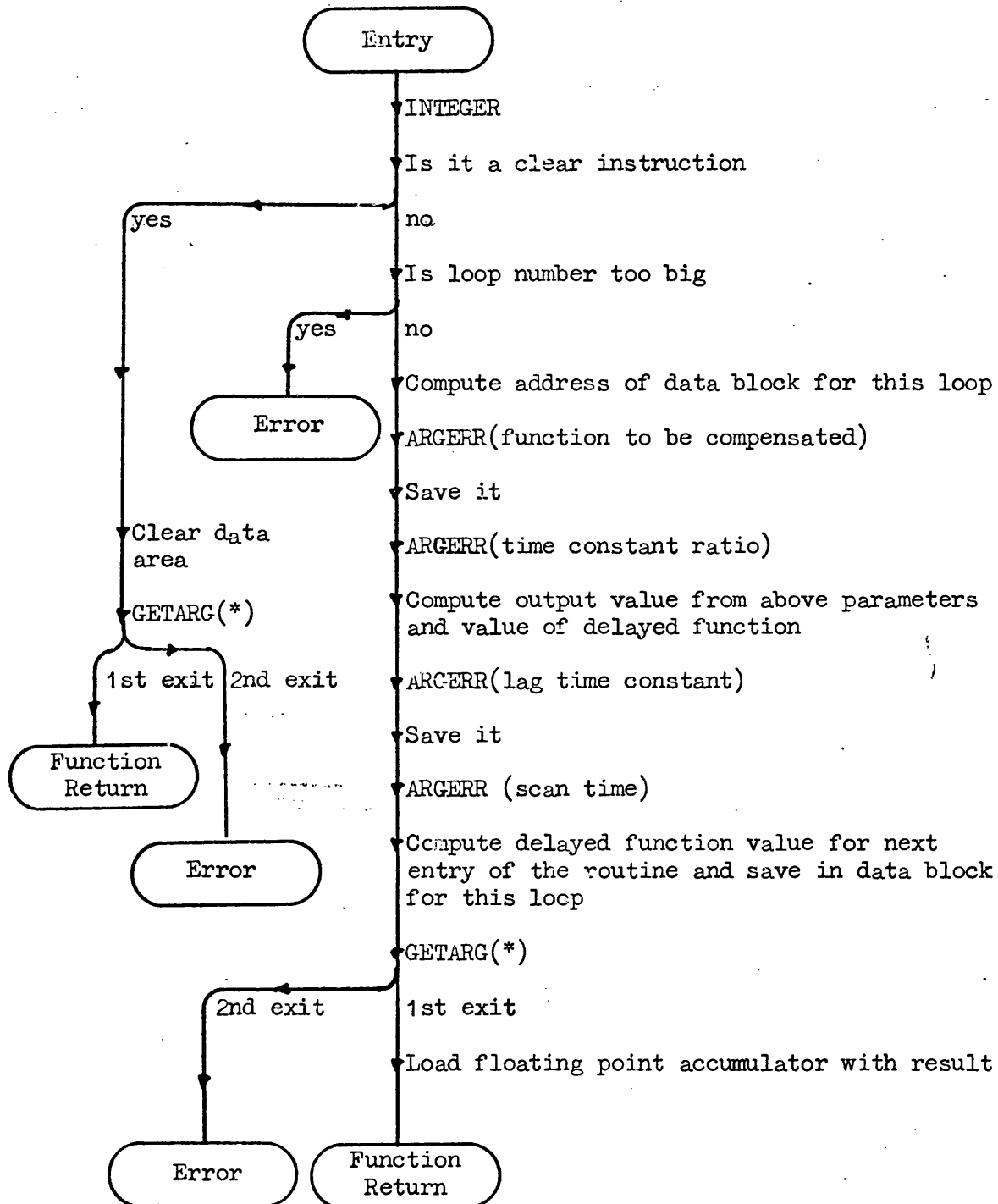
SETIT

BITSETSETAD

Output Counter Card Routine (not interrupt driven)Pseudo Random Number Generator

FCON Two term PCI control algorithm

ANGERR

Lead-Lag Compensation Function

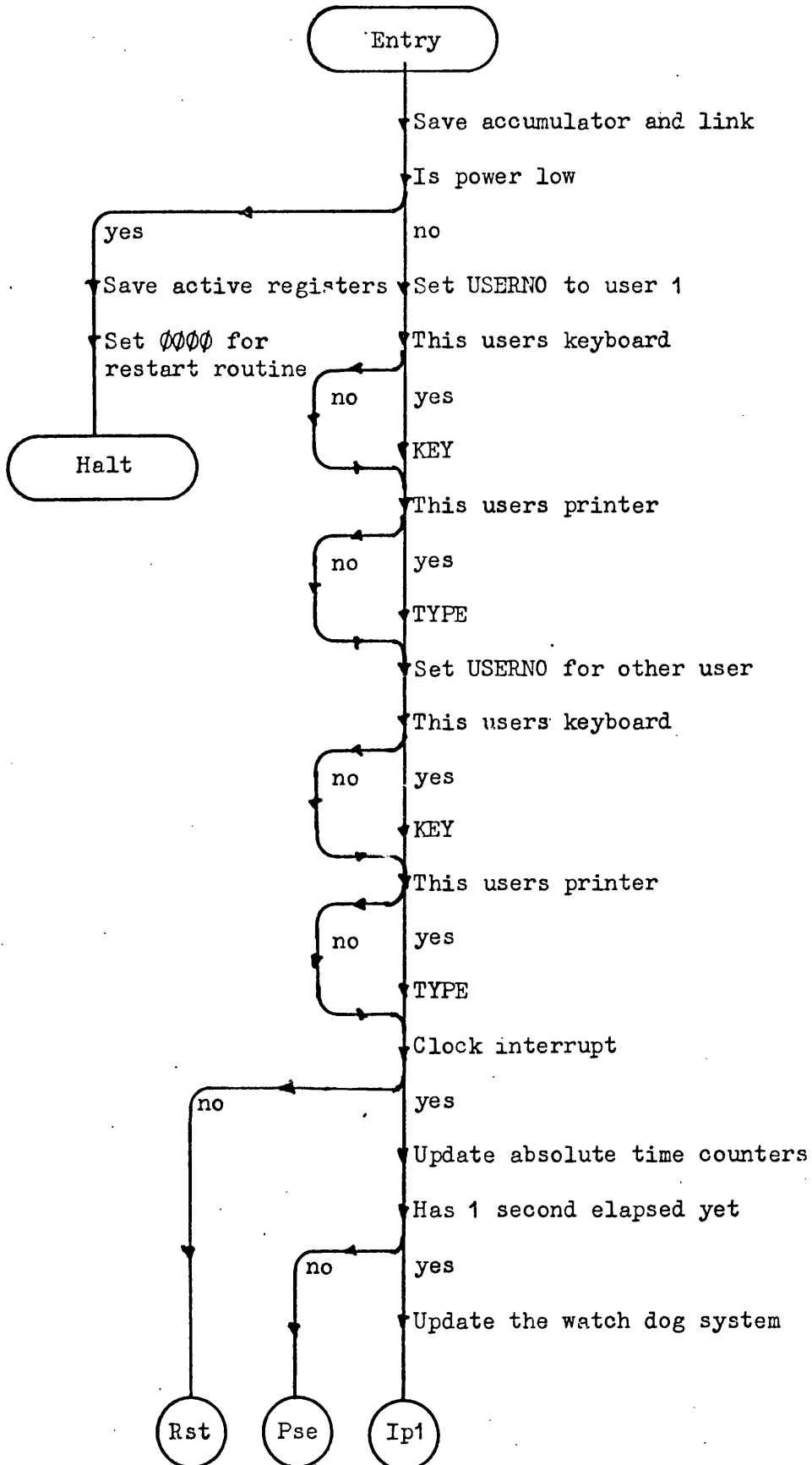
APPENDIX E.

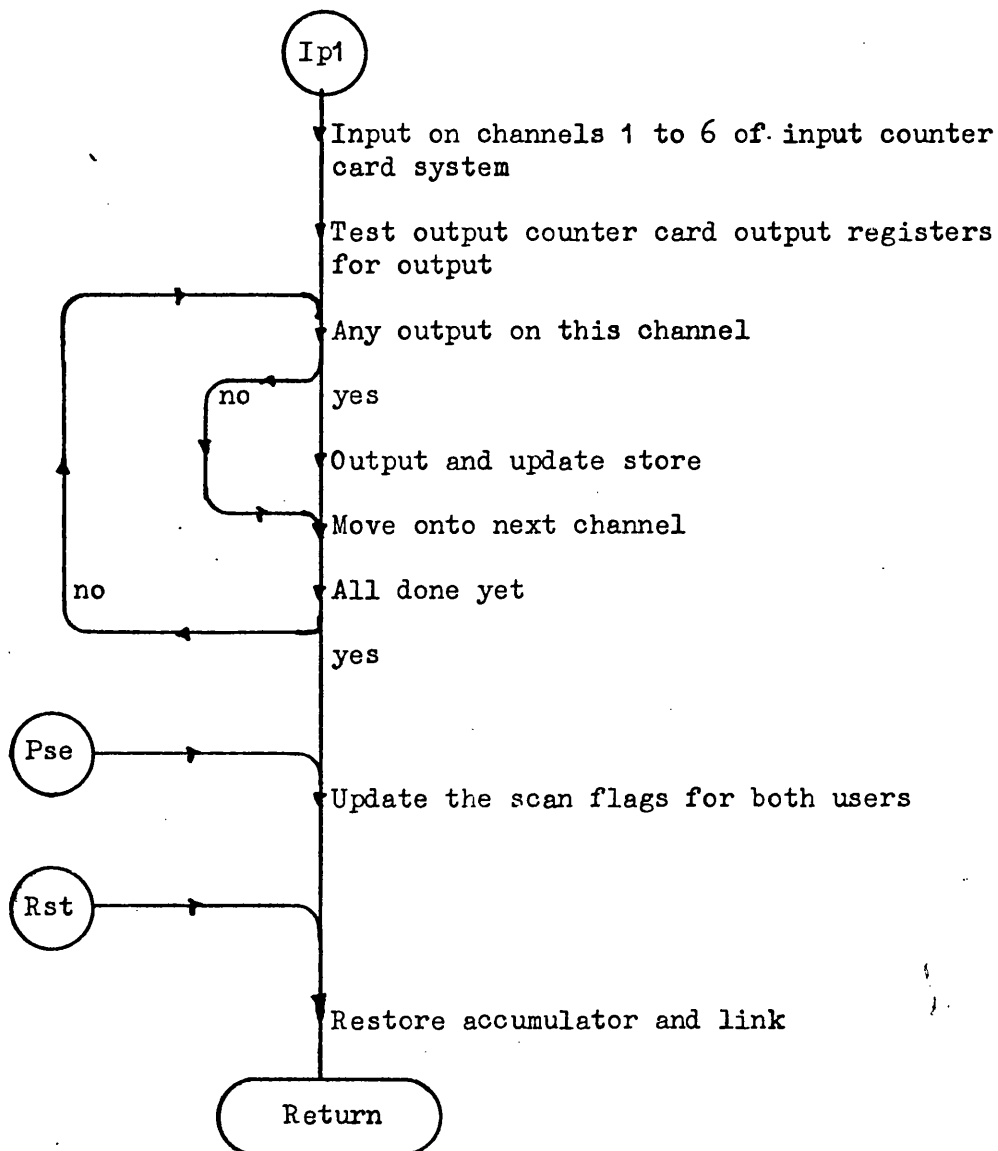
Flow Charts for Two User

Real-Time FOCAL

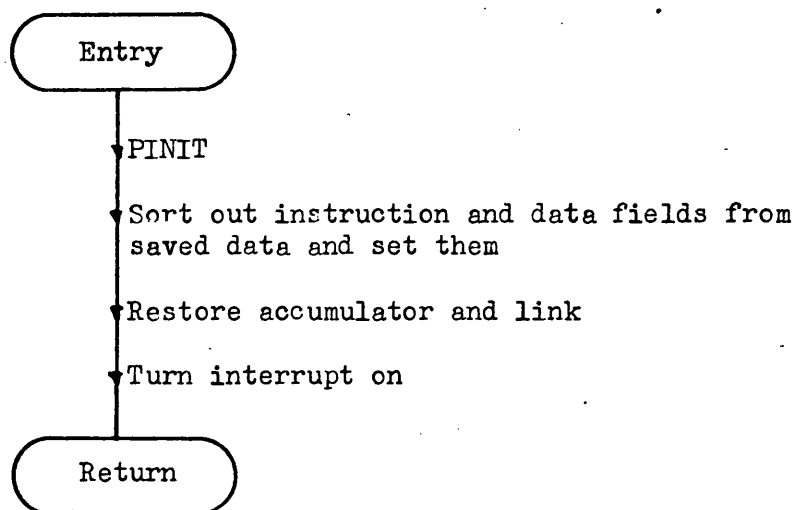
DNO Core map Field 4

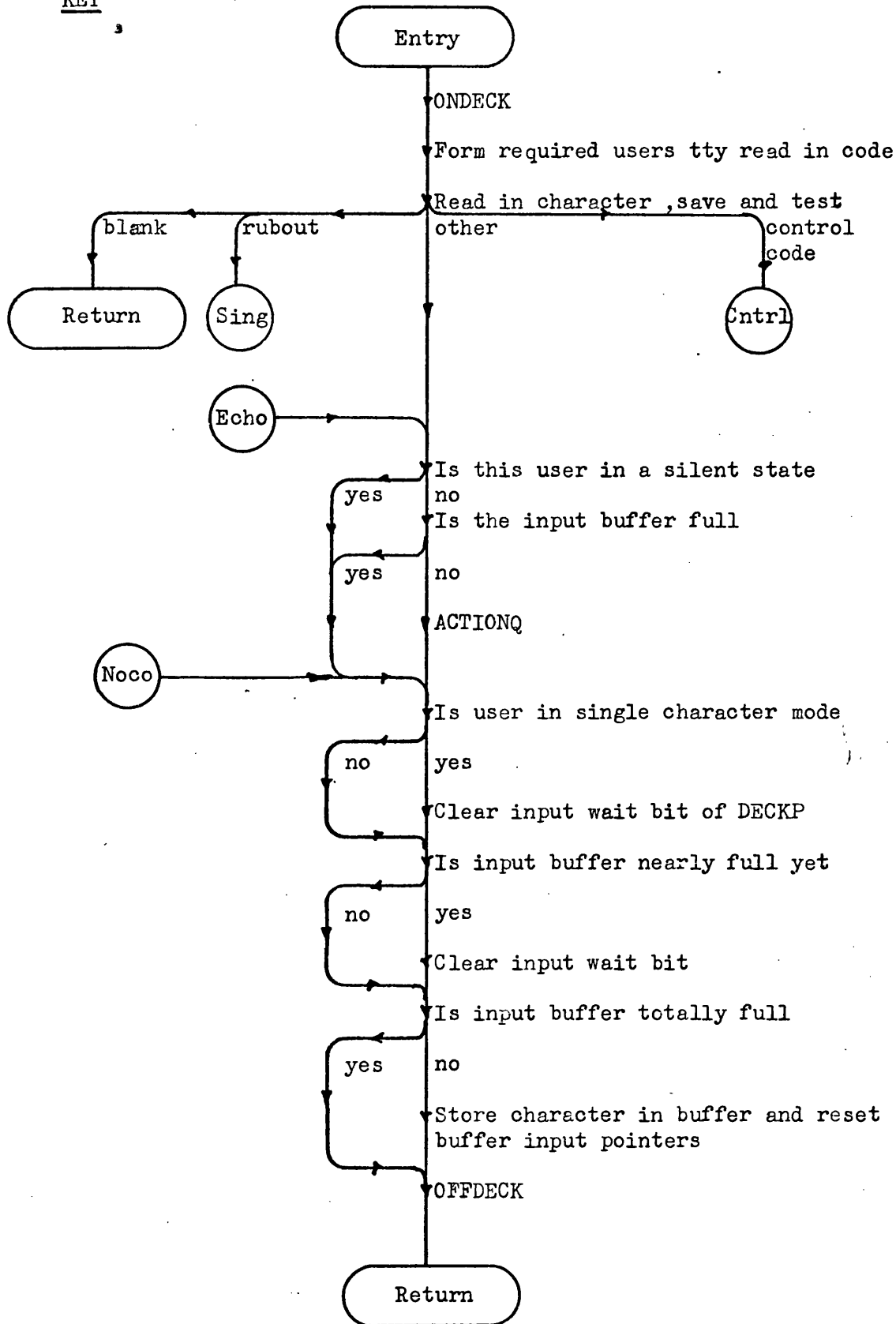
	Zero Page Pointers										
	Command Decoder				Getln						
	Fntabf		Do		Pushdown List Controls						
	Control & Transfer		Write		Testc		Sortc		Grptst Input		
	Comlst		If		Set & For		Dys & Int		Congo		
	Type & Ask		Modify		Sortj		Adc, Outl and others				
	Getarg				Spnor, Testn, Ran, Popj and others						
	Ecall										
	Terms Sgn, Abs et al		Tstlpr, Partest		Delete		Readc		Fntabl		
	Erase		Findln		Geto		Endln		Extensions		
	I33, Prntln, Prnt and Printc				Packs						
	I/O Routines, Swapping Routine, Mods to Ask & Modify, Single, Sortnset, Error Recovery, Adtst				Extensions, Psymb8, Gsym8, Gsub8						
	Rubout				Symbol Table Dump		Variable Erase				
	Extensions to Library and Variable Search Routines										
	Parameter Modification Routine and Dipchk Routine										
	Argument Evaluation		Flag		Ftim		Lists		S.t.d Extn		
	PCI Control Algorithm										
	Fin and Fout Commands										
	Fran		Exp		PCI Store Table		Atn				
					Log						
	Cos and Sin				Mod. and Altmode						
	FLOATING POINT										
	INPUT AND OUTPUT										
	ROUTINES										
					E C and E D						
	FLOATING POINT INTERPRETER										
	Sgt		Library								
			Loaders								

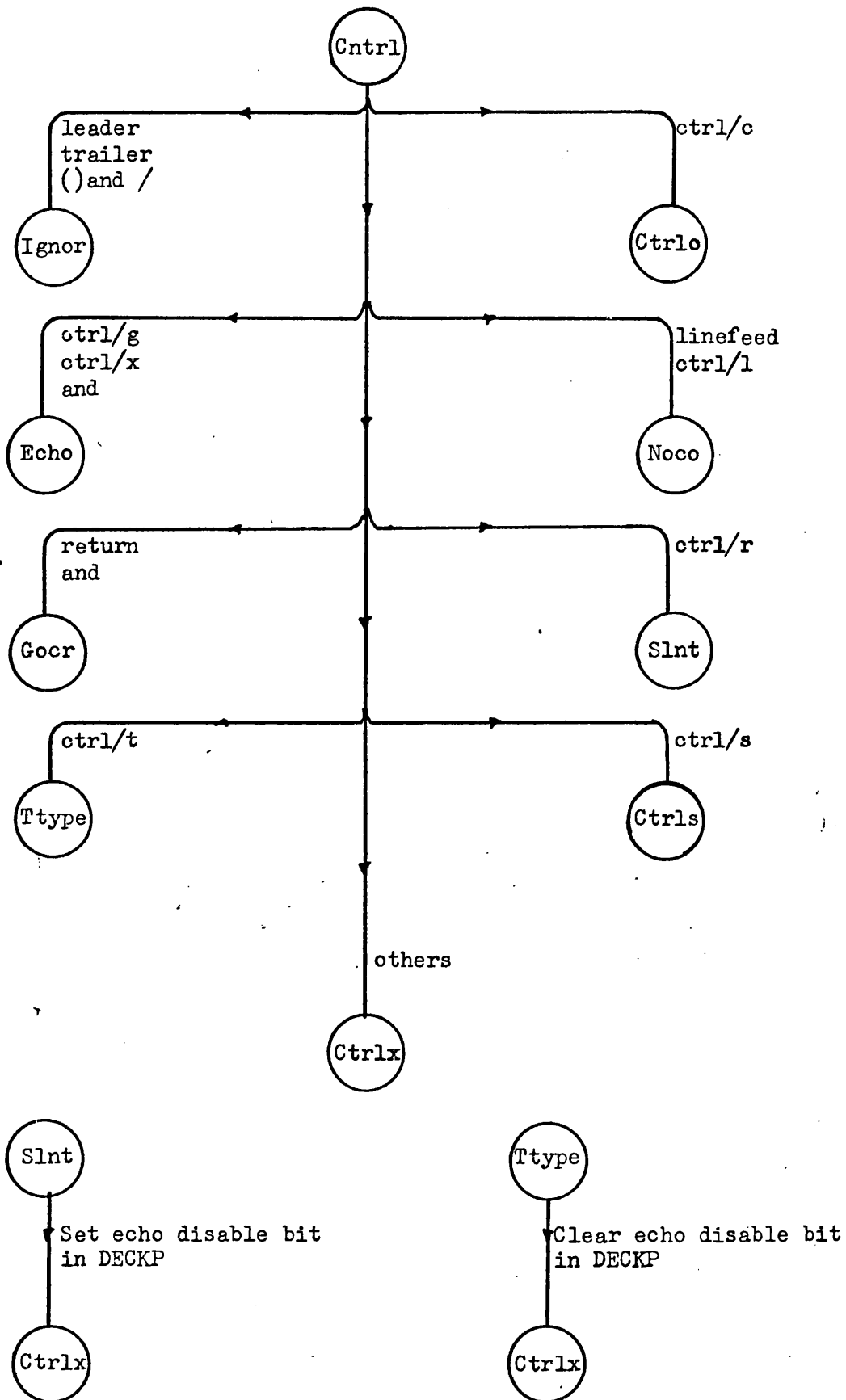
Dynamic Duo Interrupt processor

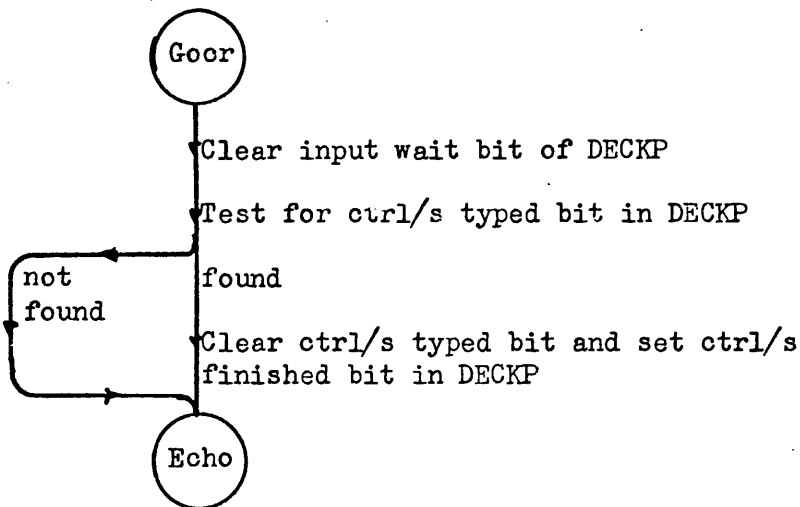
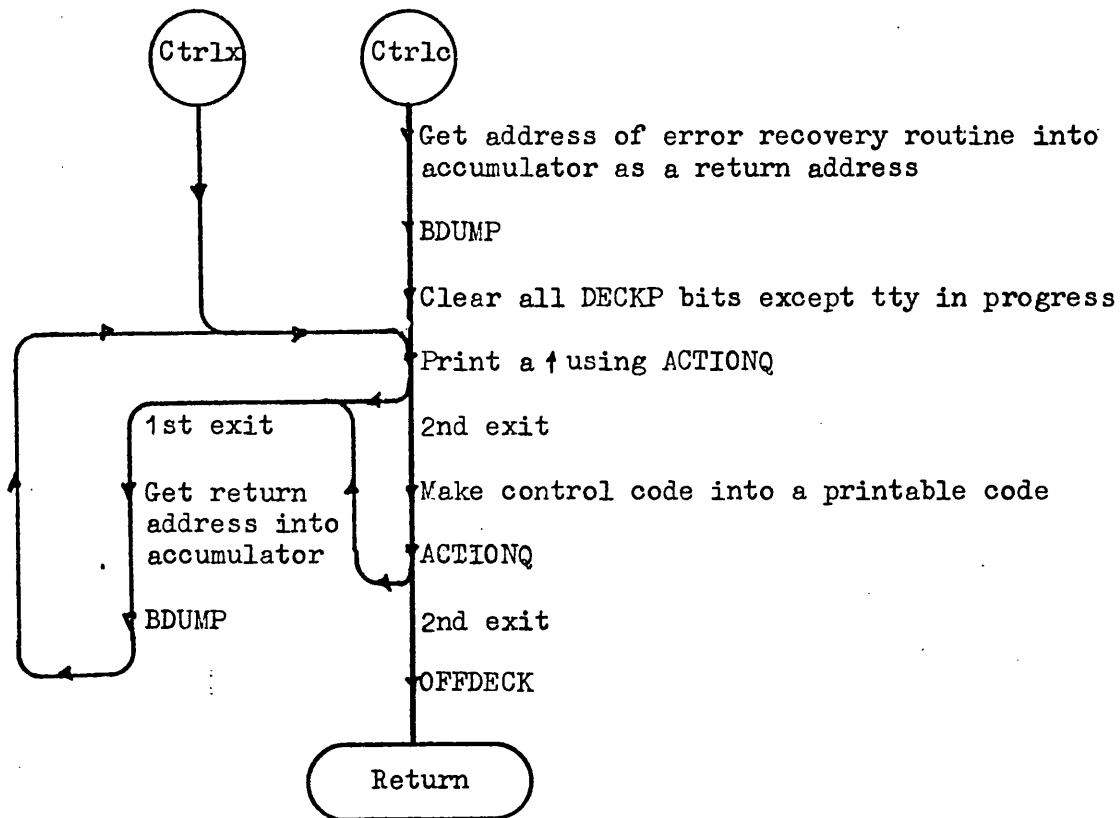


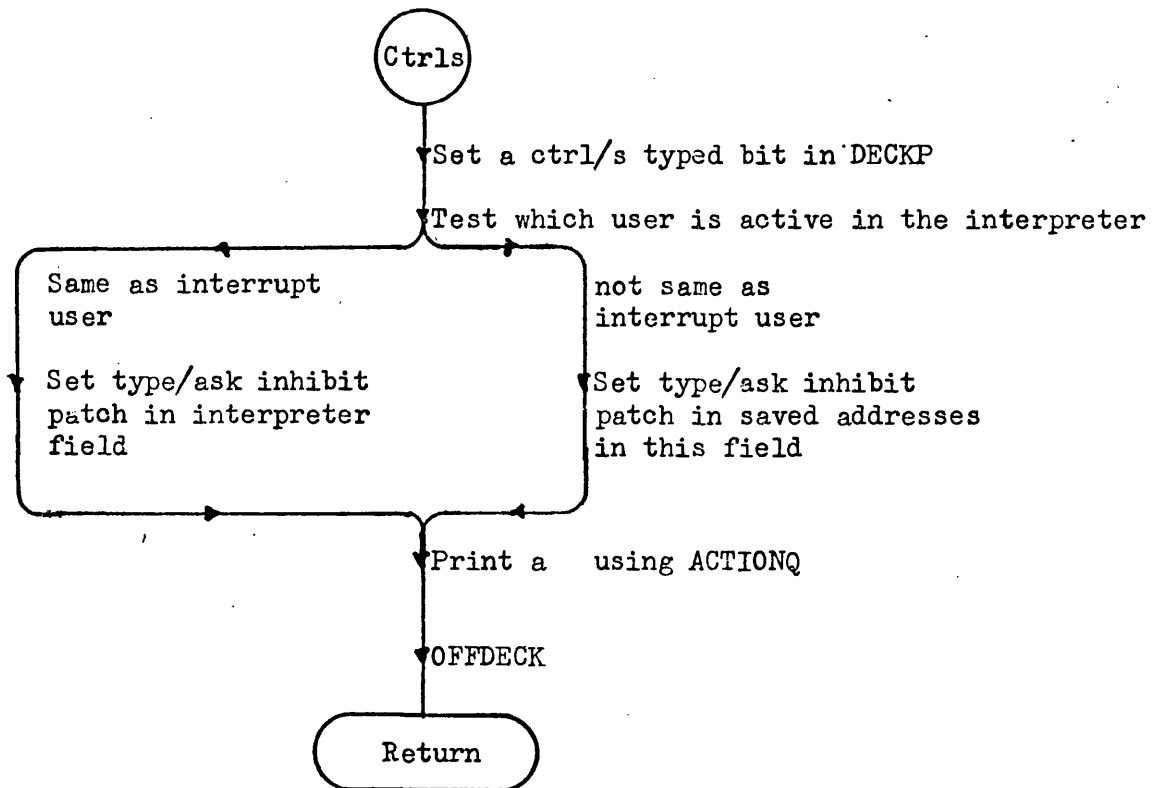
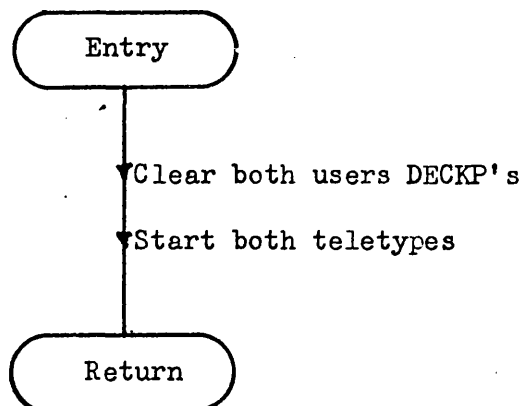
Power On Automatic Restart



KEY

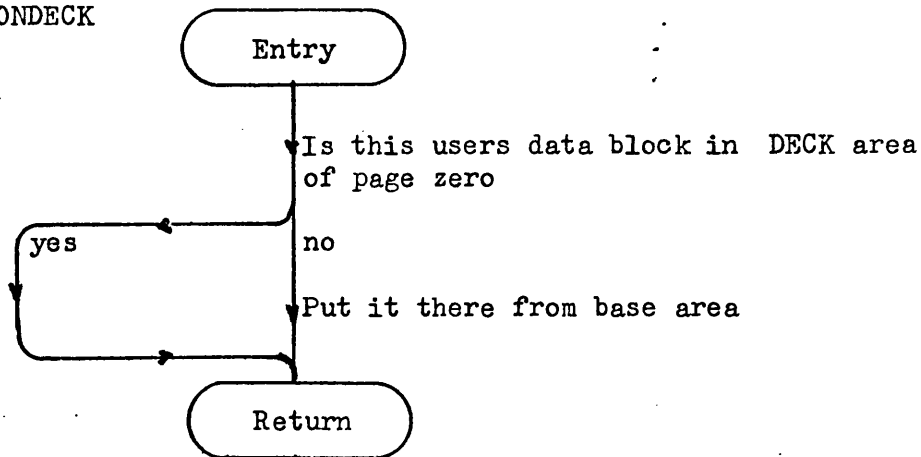




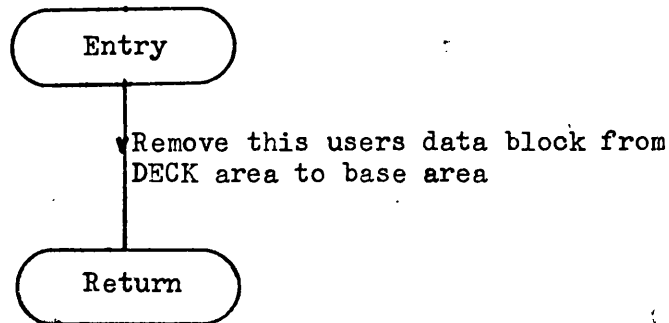
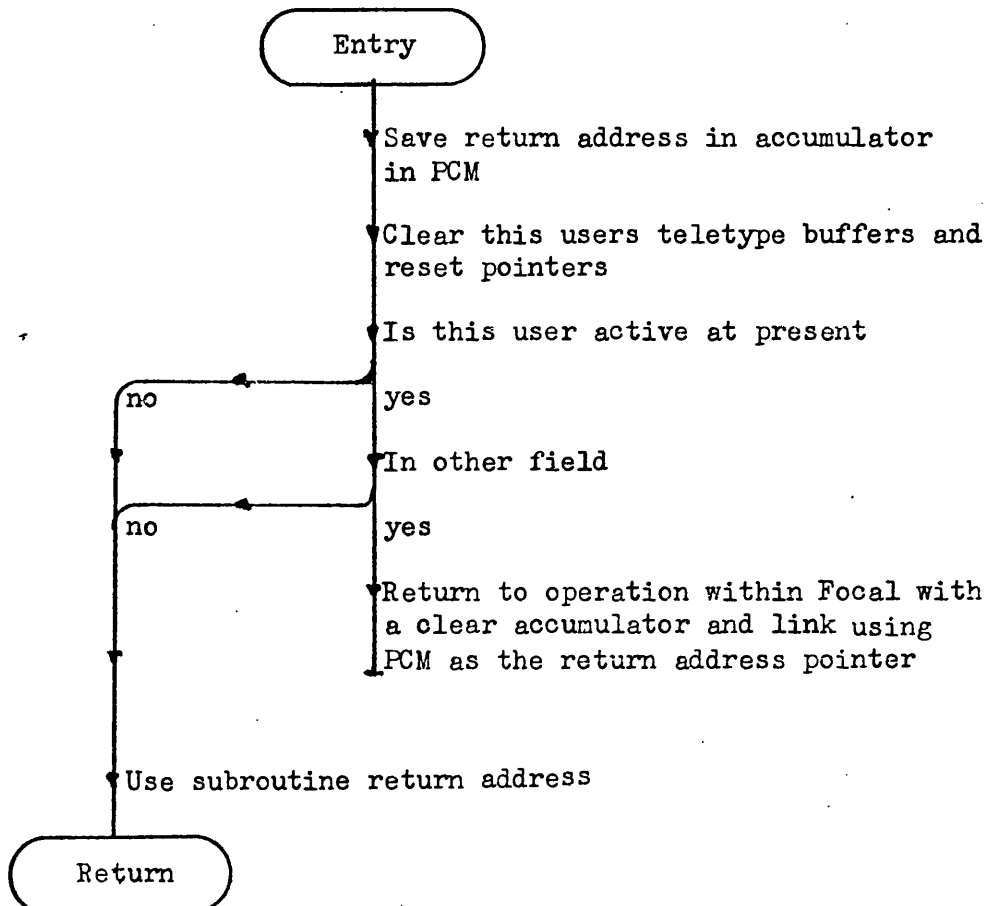
PINIT

XDECK

Called by ONDECK

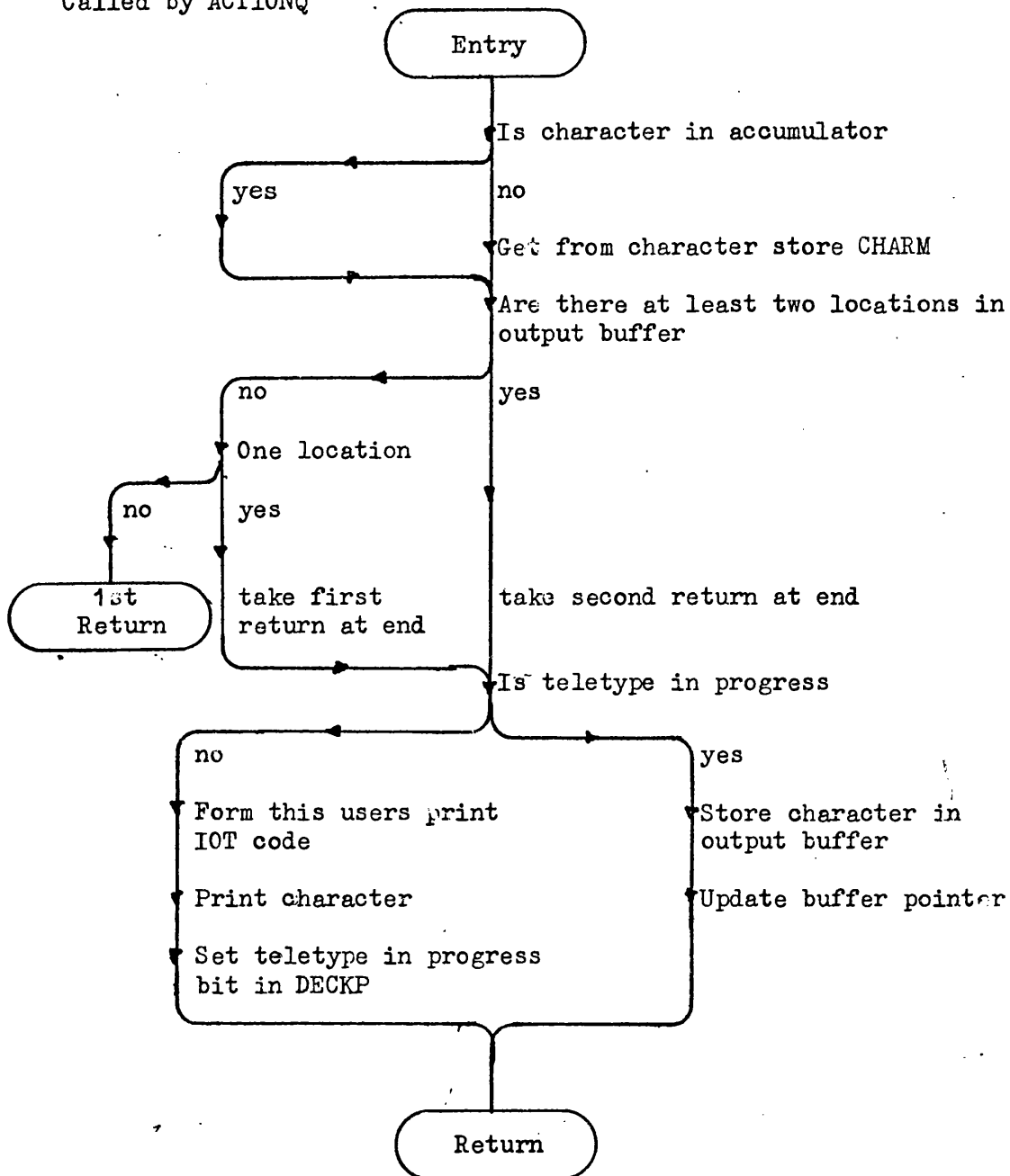
UNDECK

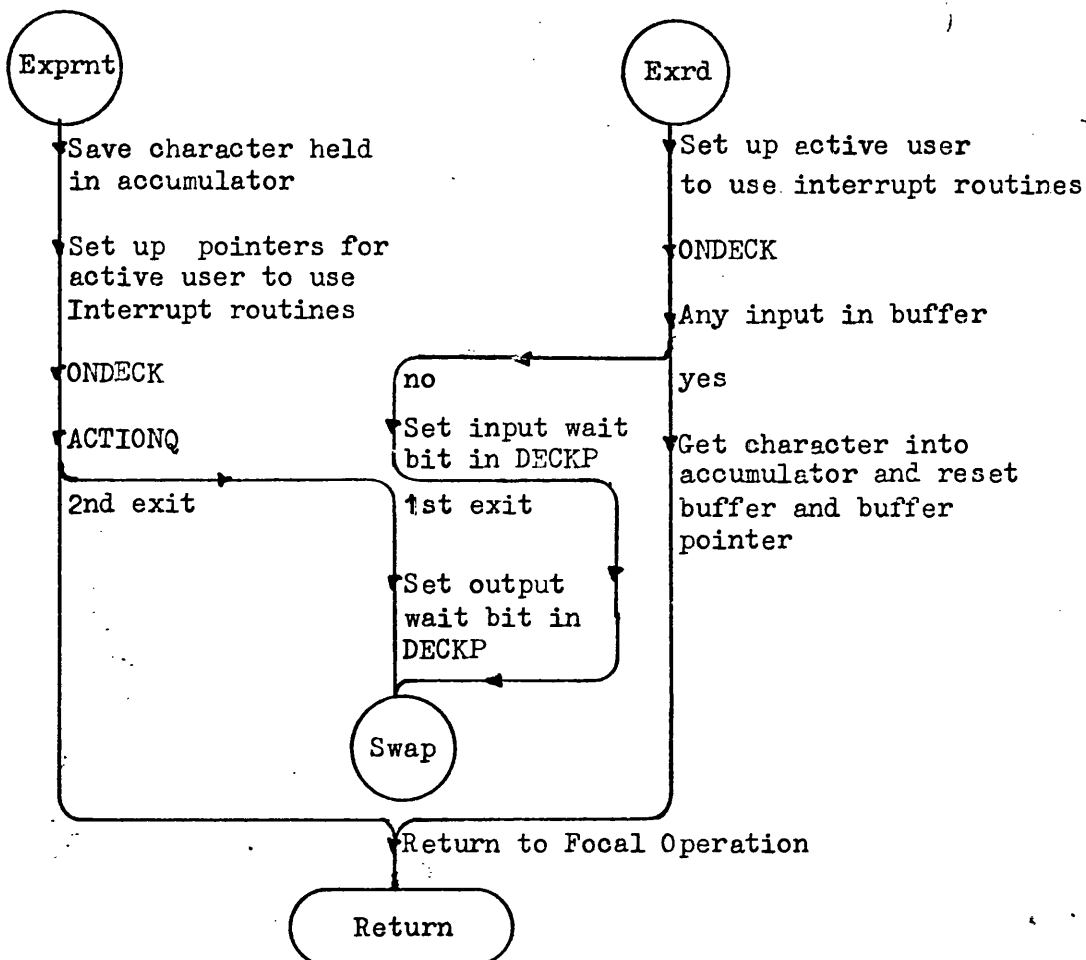
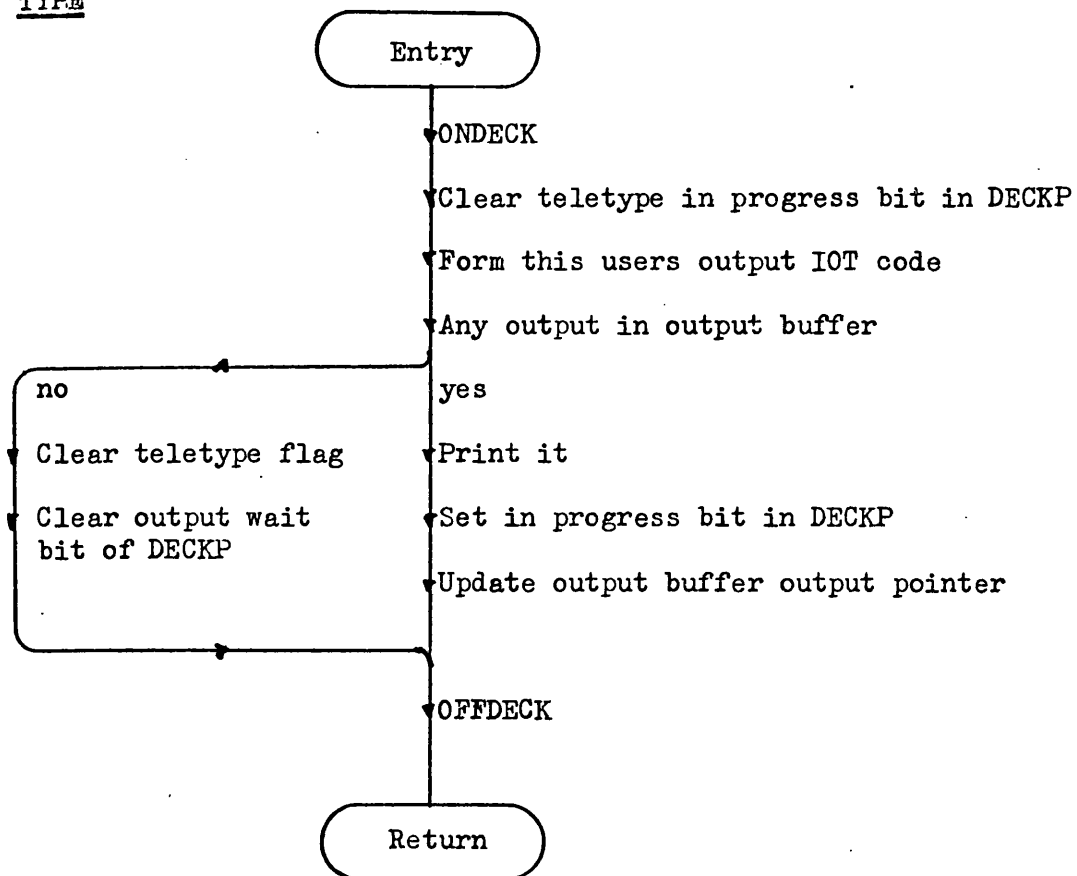
Called by OFFDECK

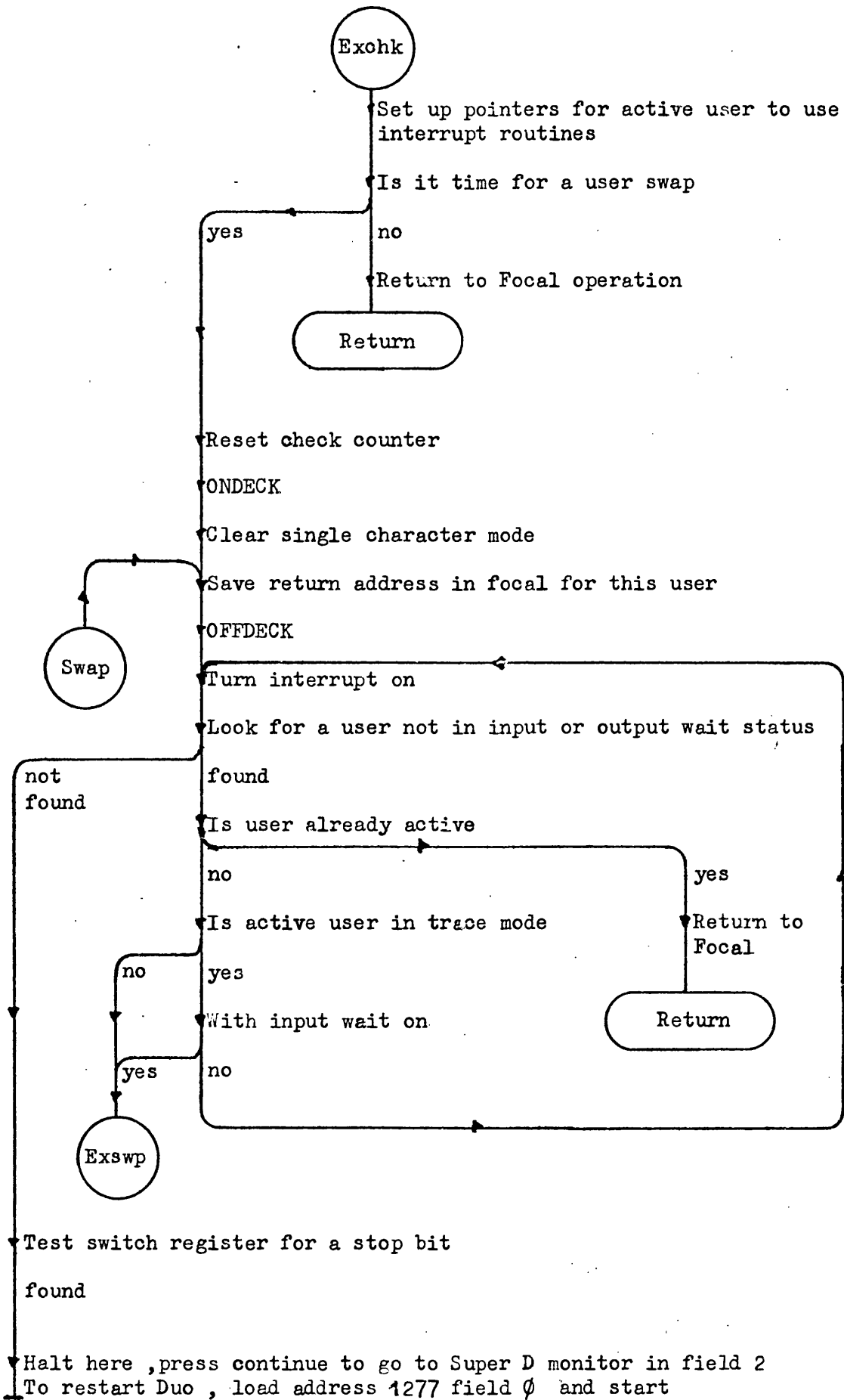
BDUMP

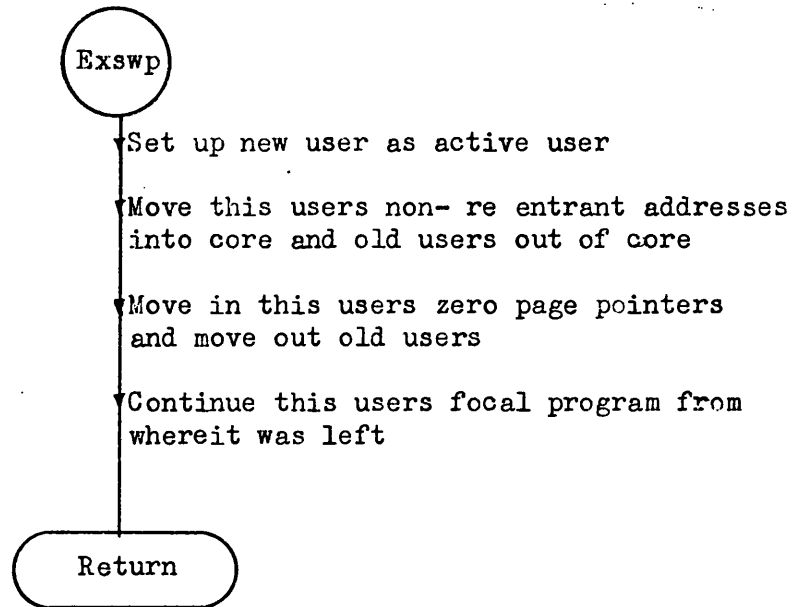
XACTION

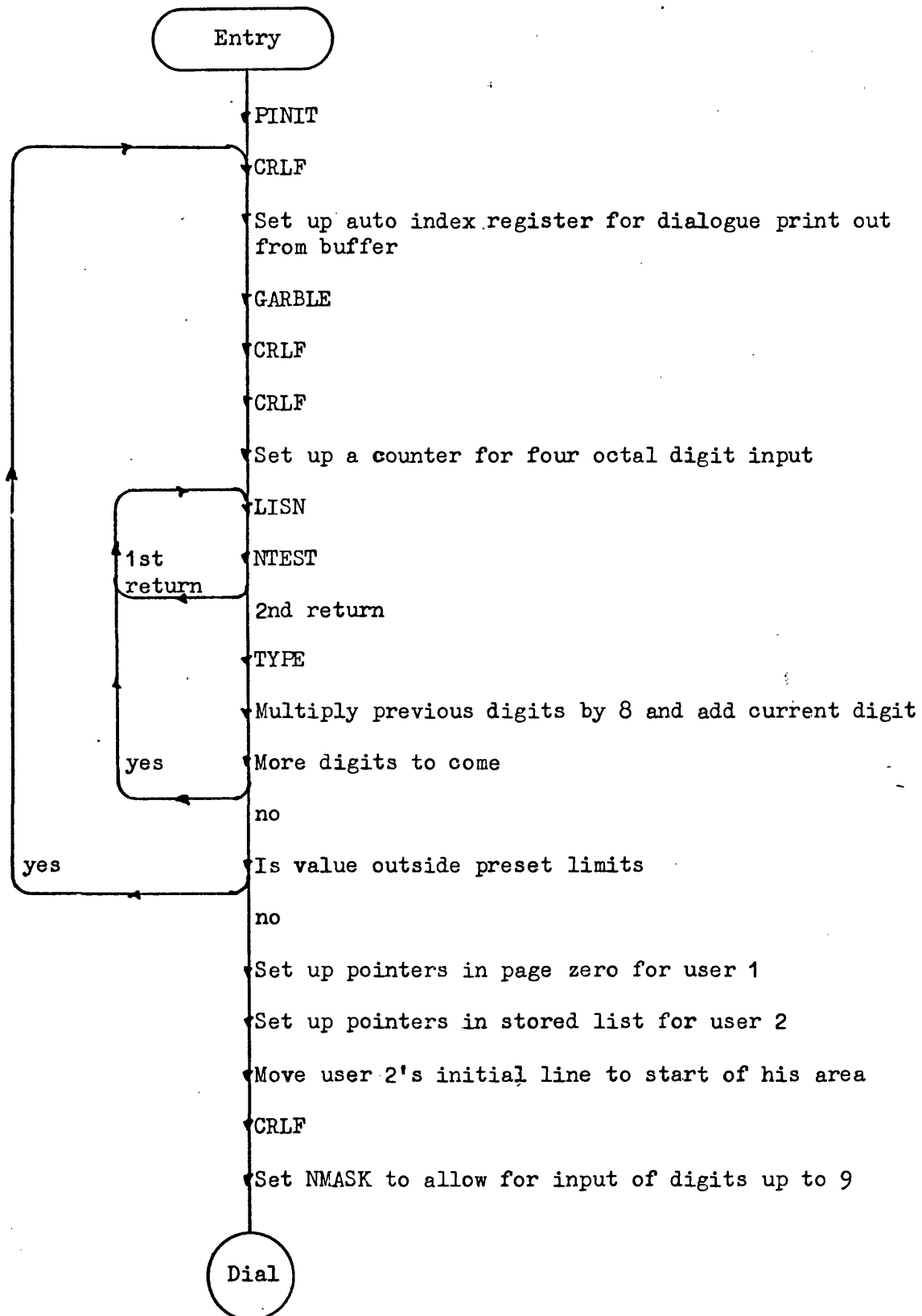
Called by ACTIONQ

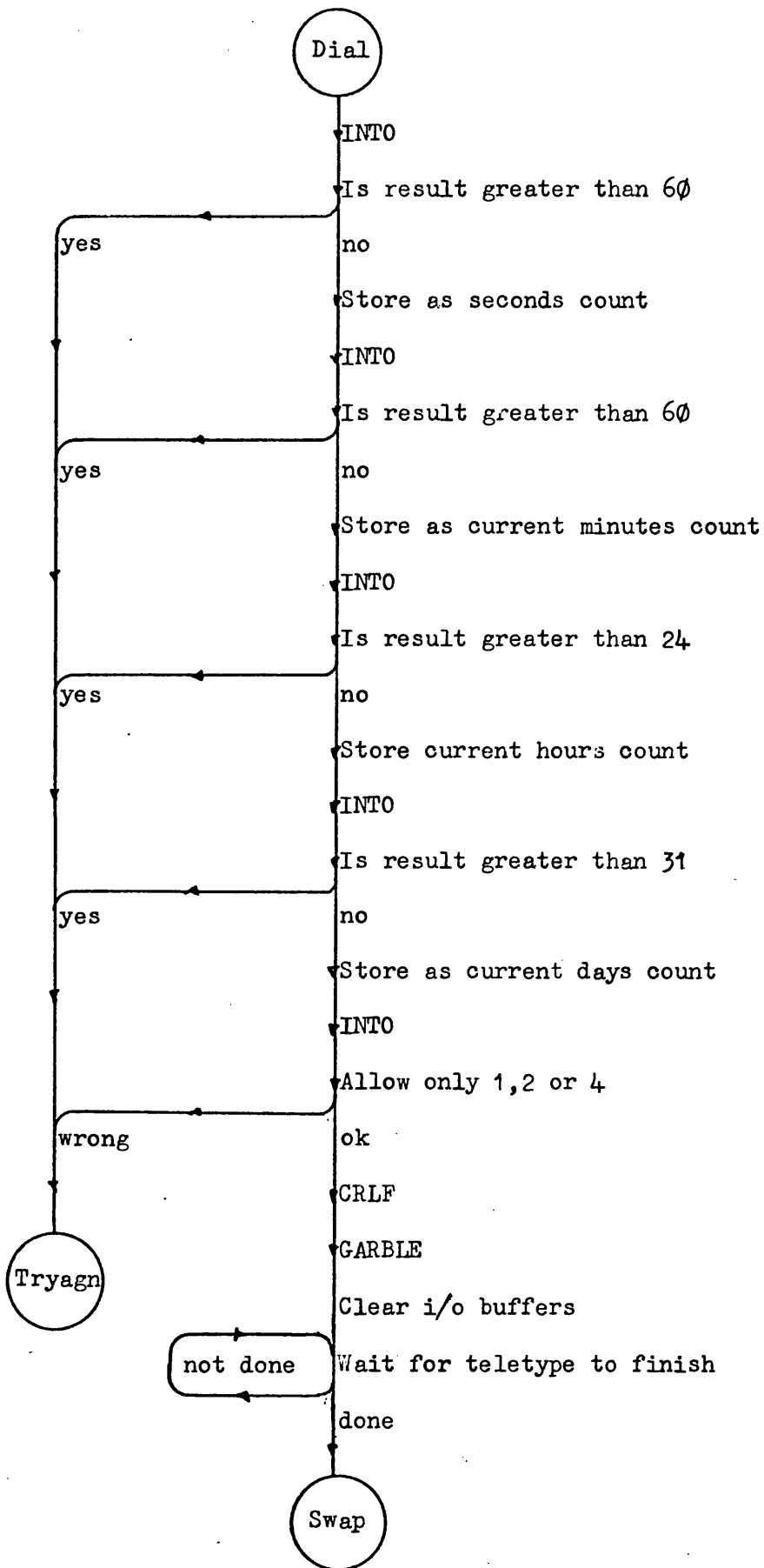


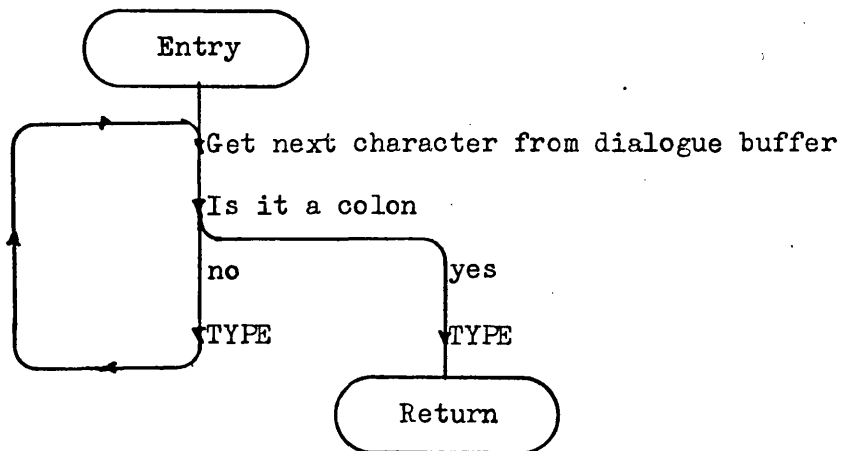
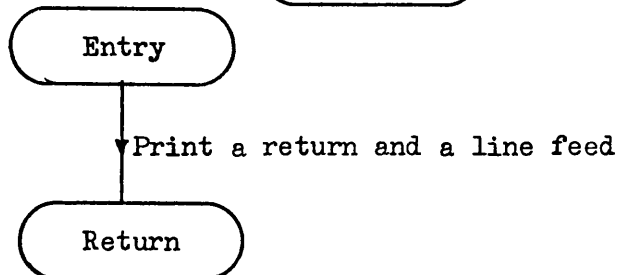
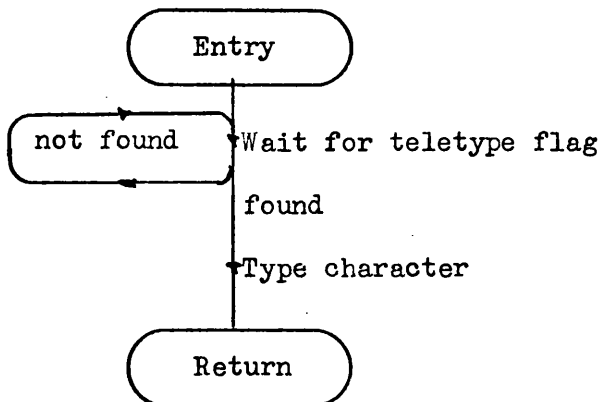
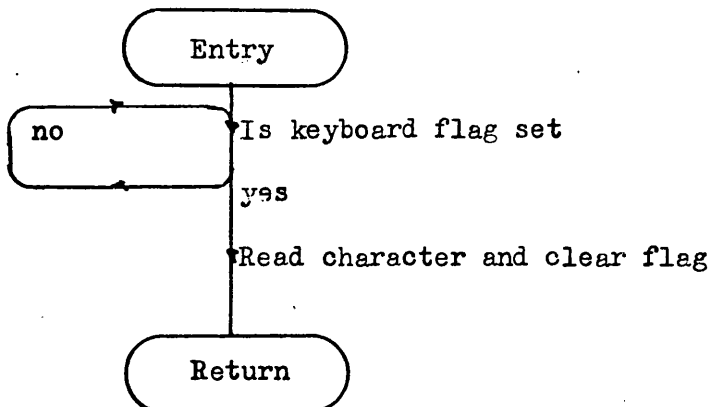
TYPE

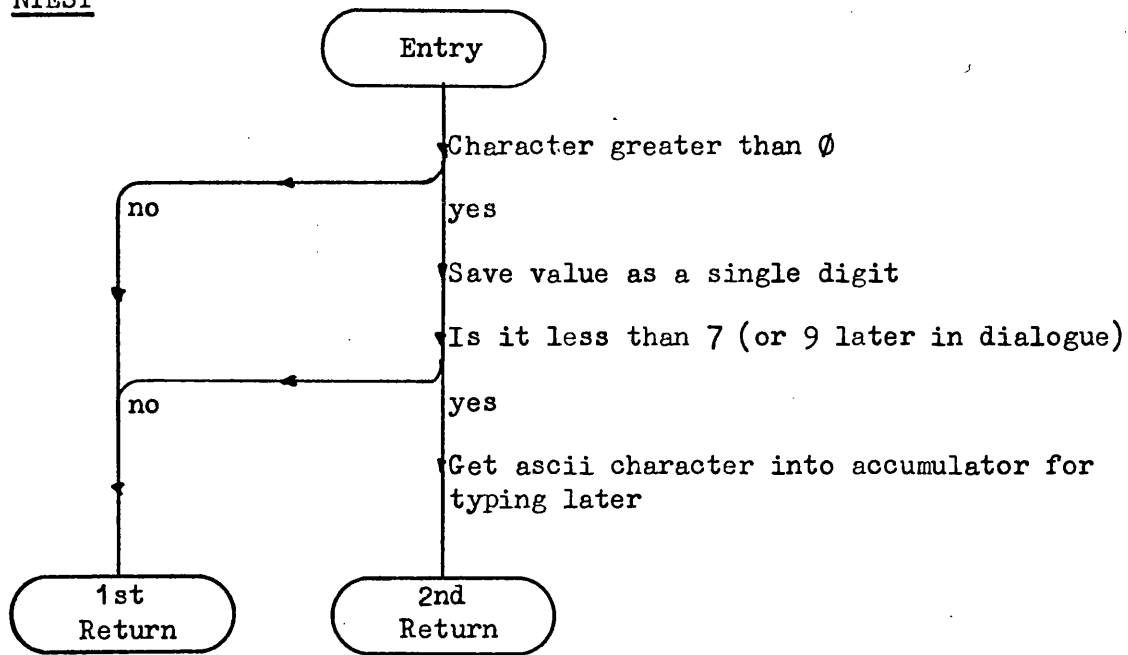
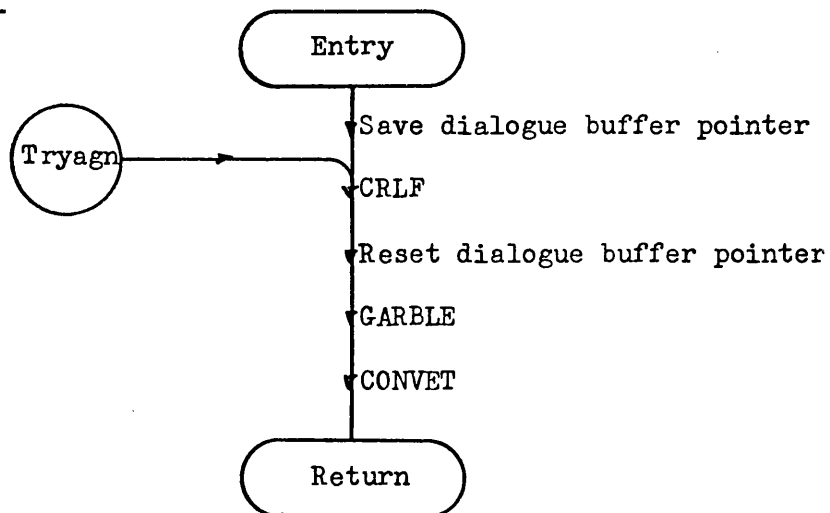


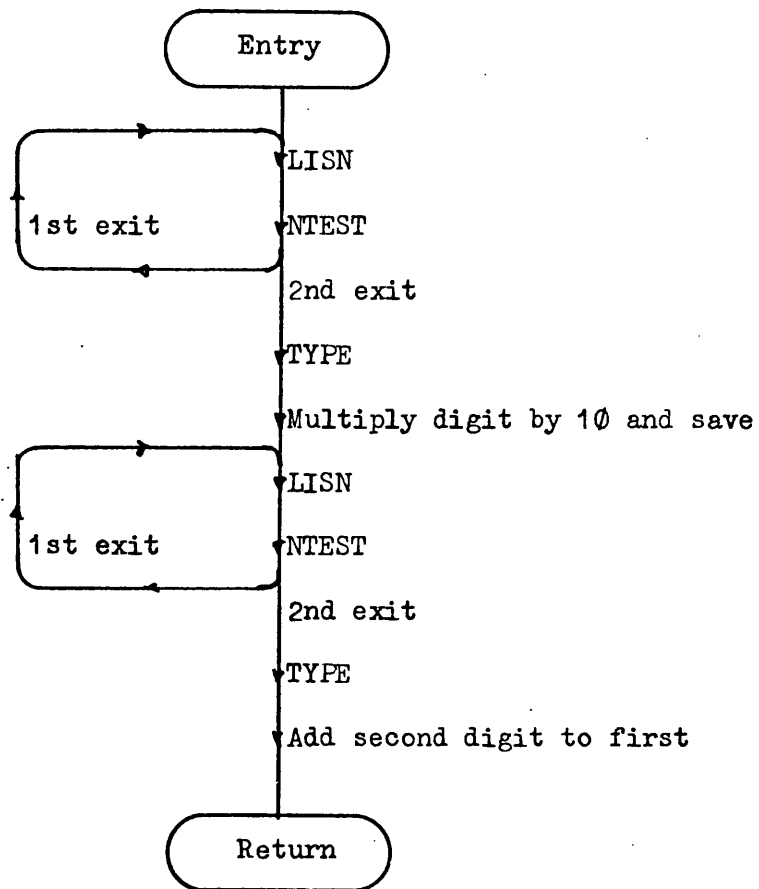


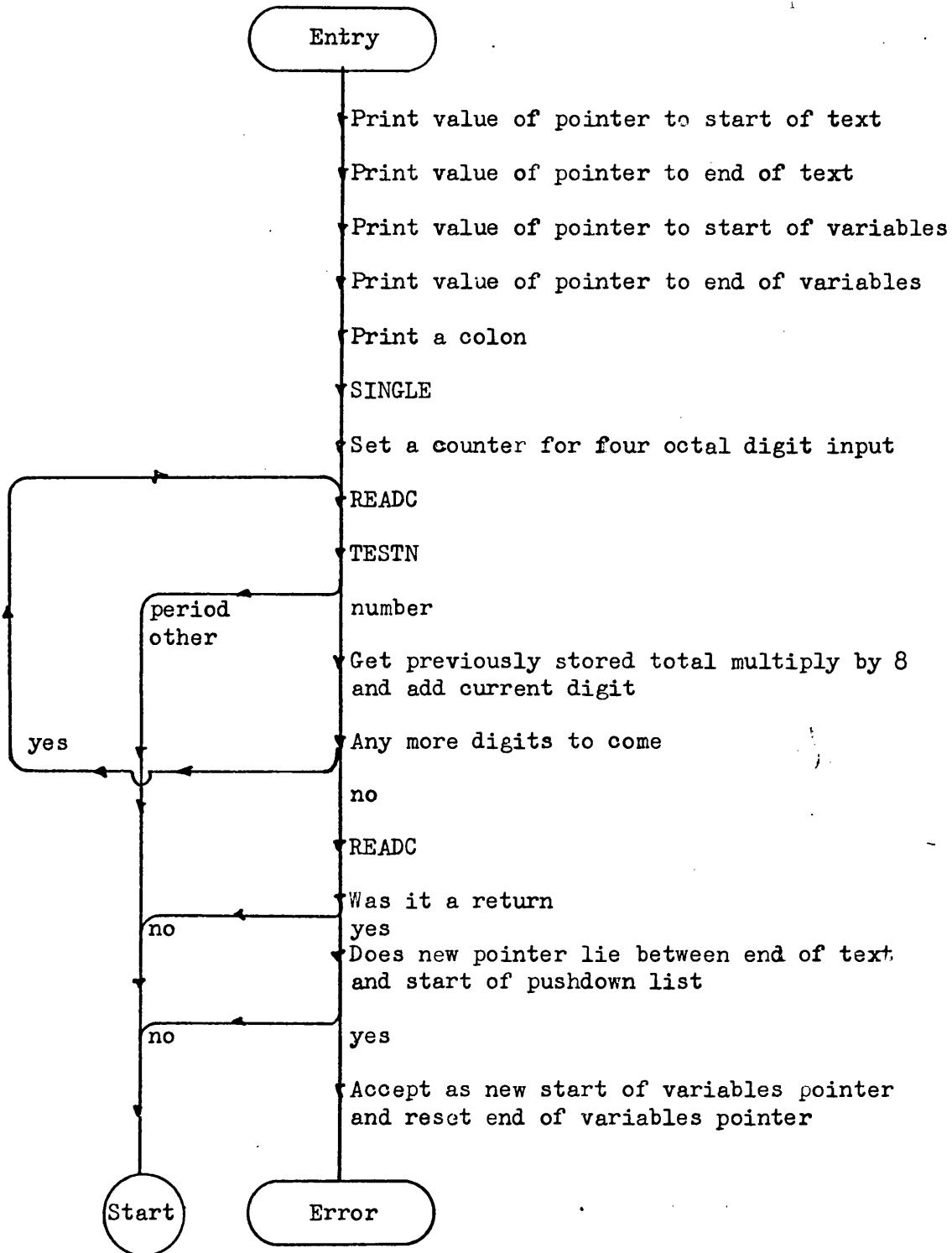
Initial Dialogue for Duo



GARBLECRLFTYPELISN

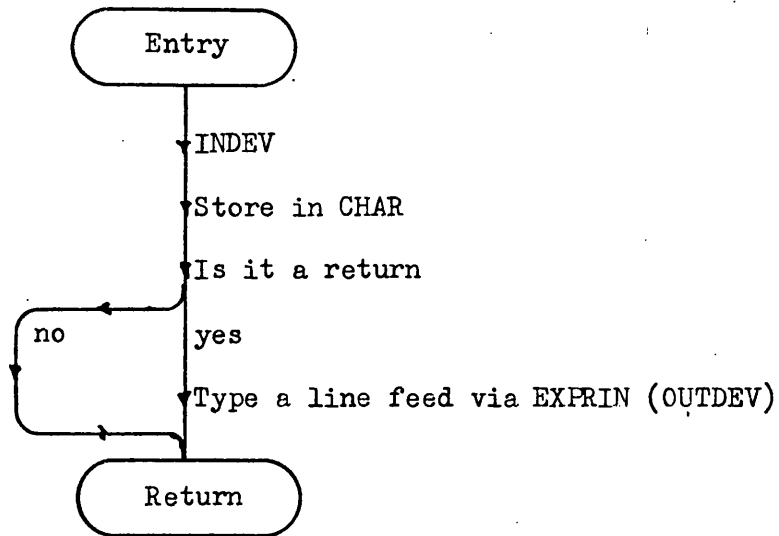
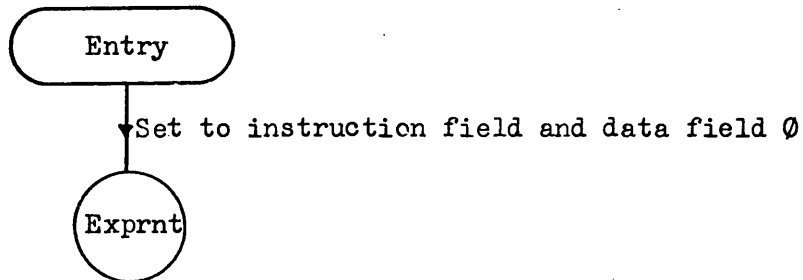
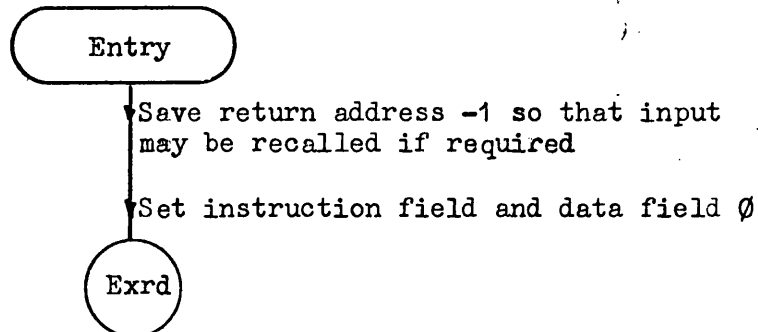
NTESTINTO

CONVET

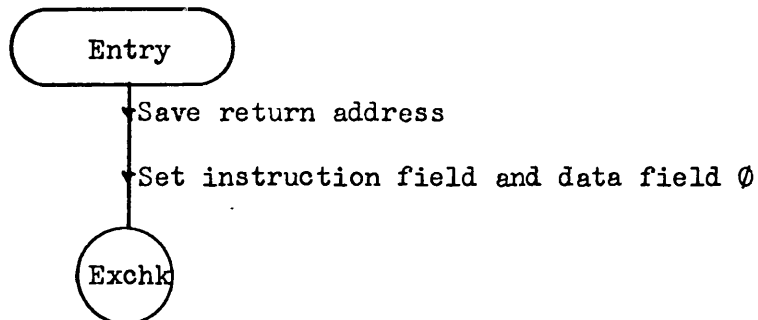
Library Modification

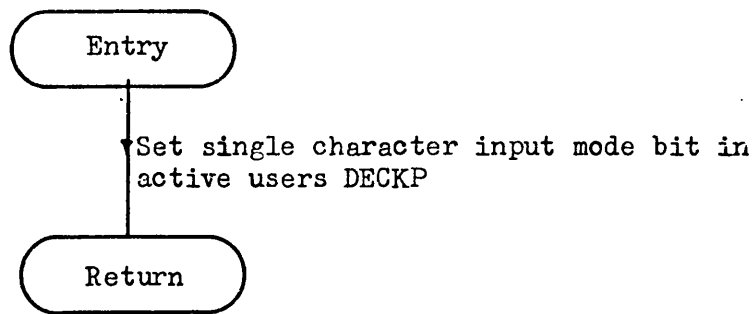
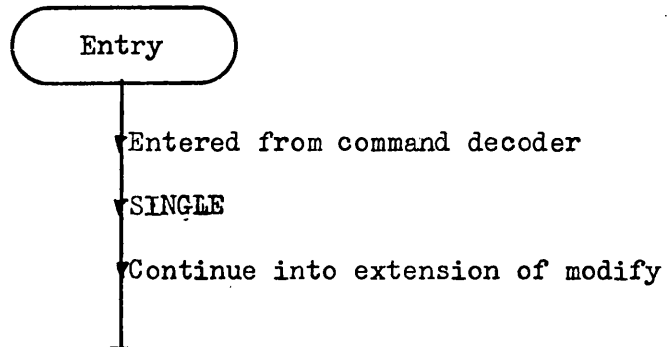
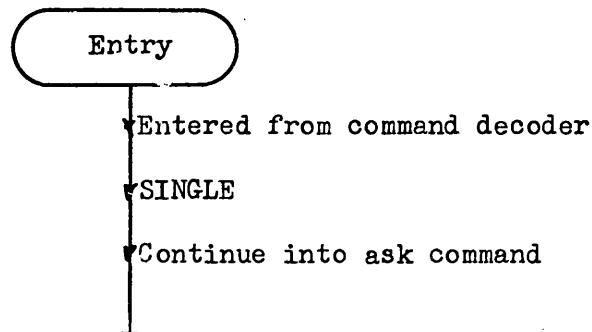
CHIN

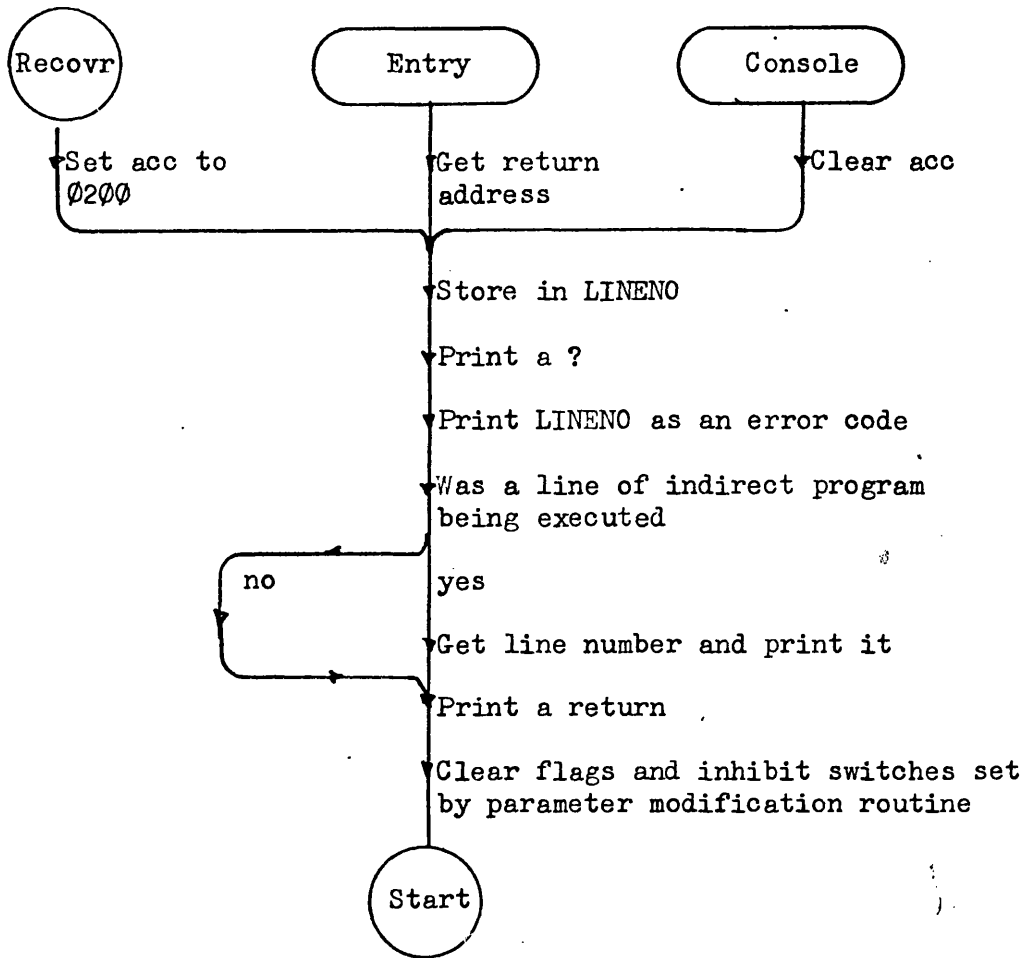
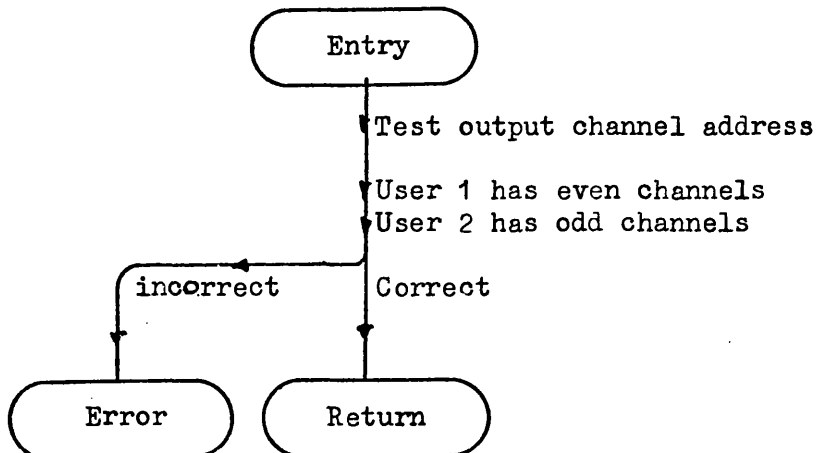
Called by READC

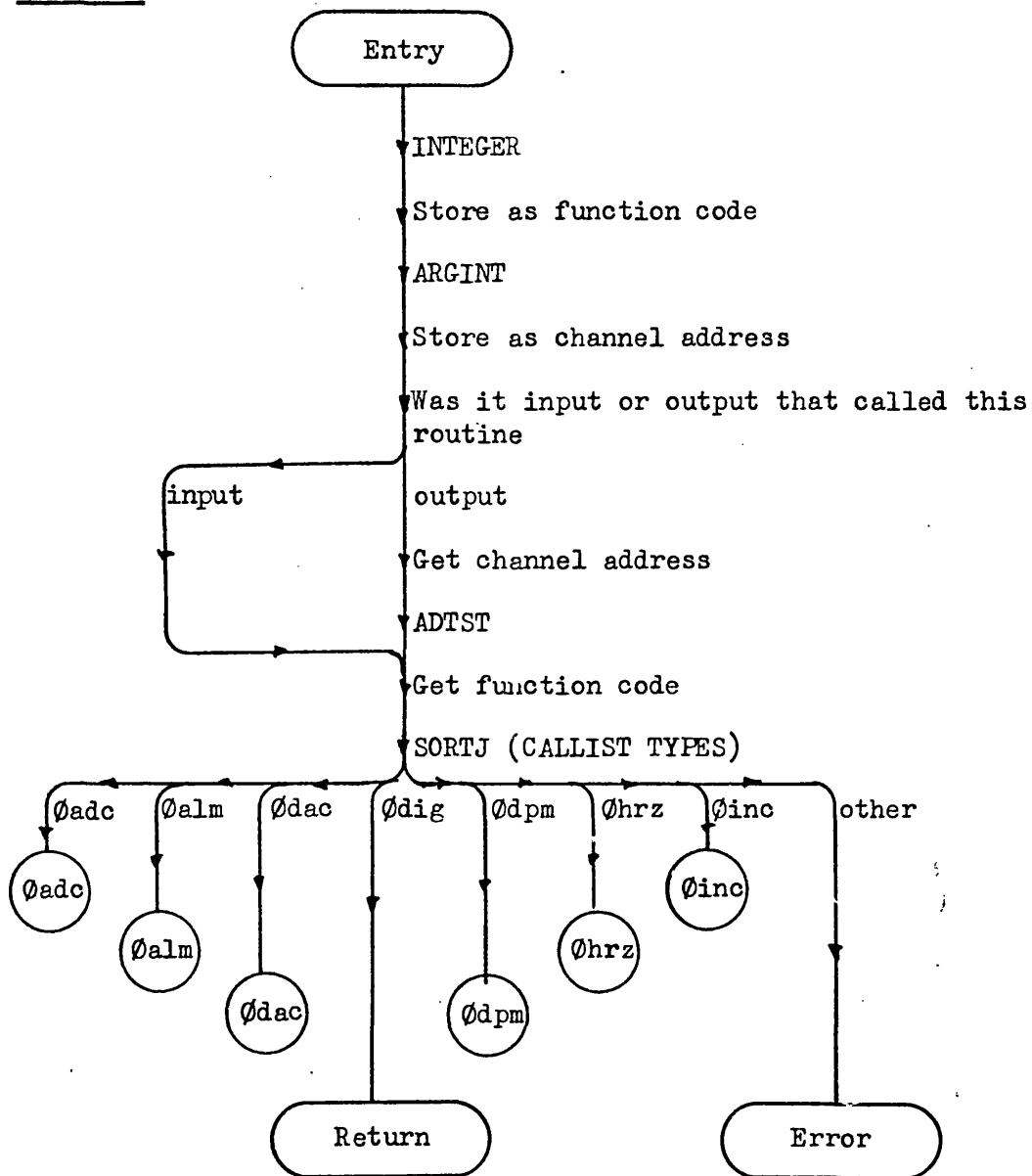
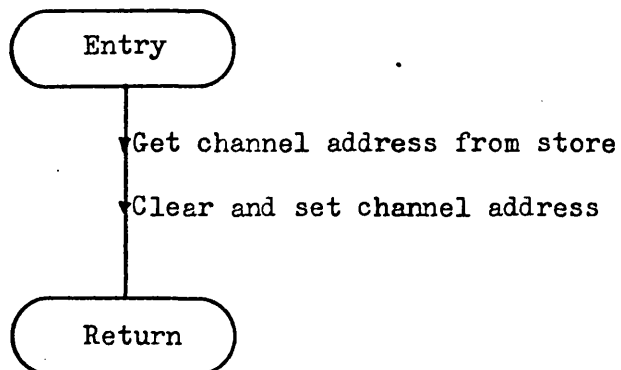
OUTDEVINDEVEXCHEC

Entered at the start of every sub line as a swap check point



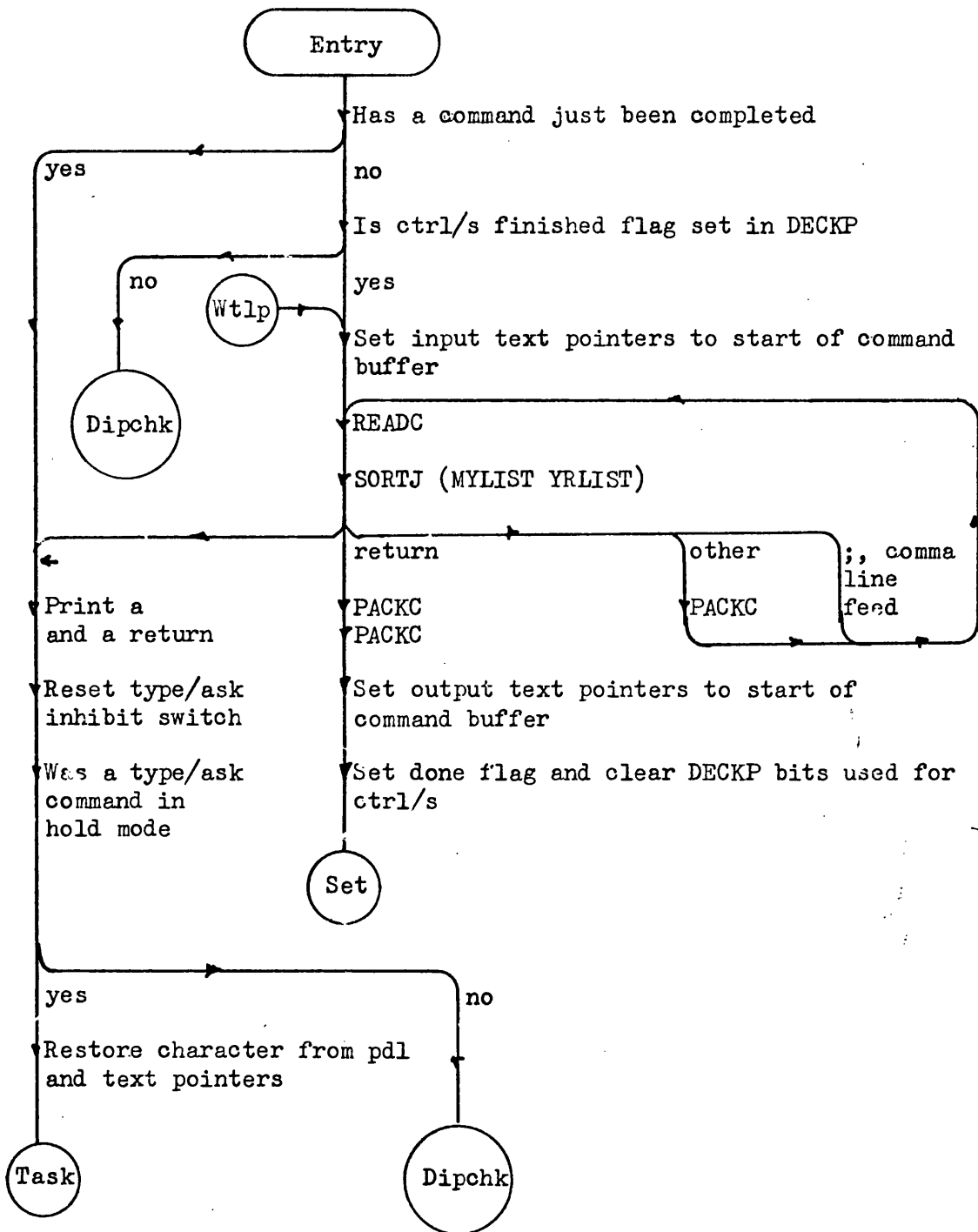
SINGLEModification to Modify Command for use with DuoModification to Ask Command for use with Duo

New Error Recovery RoutineAD1ST

SORTNSETSETADD

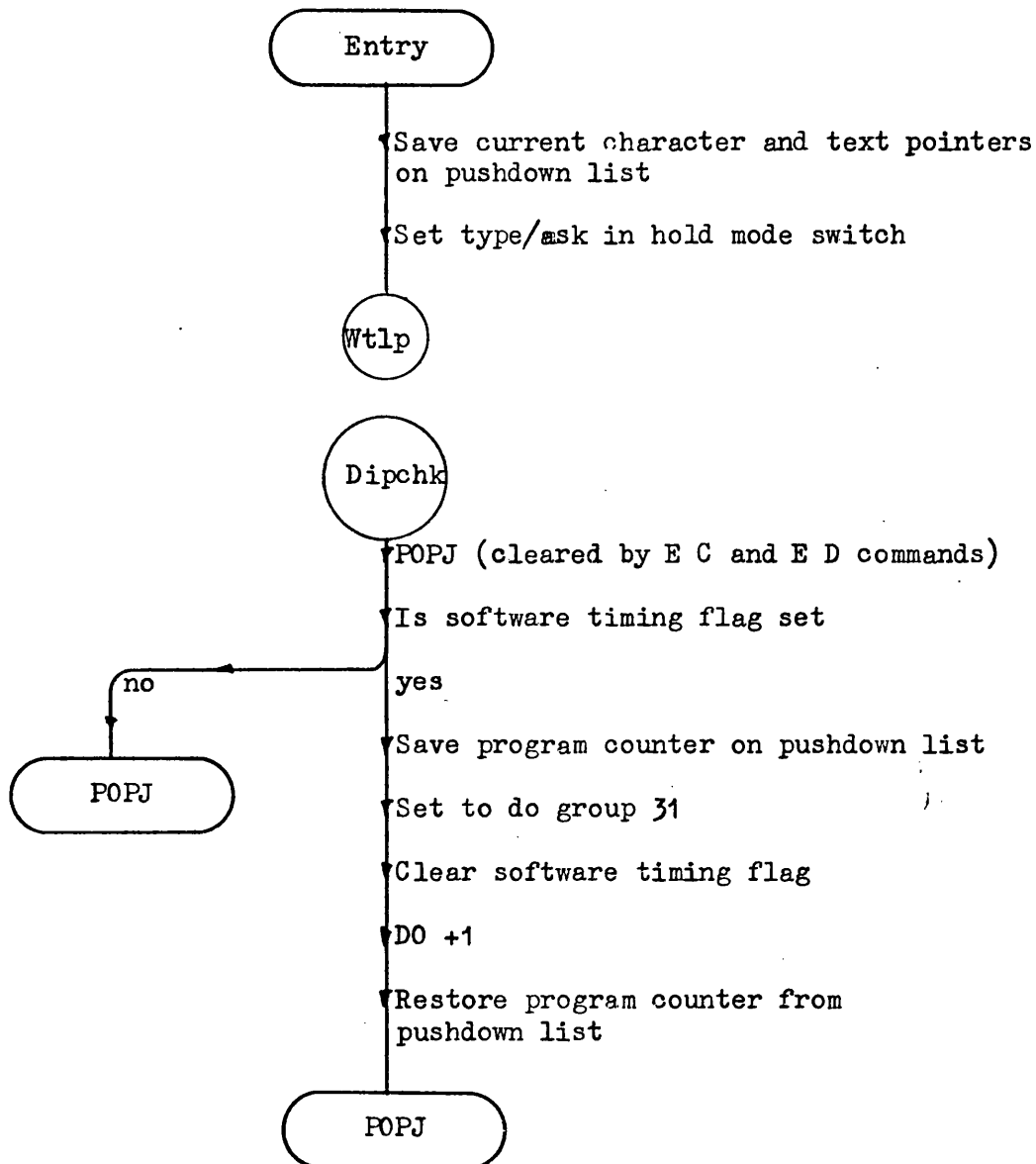
Parameter Modification Routine

Entered at the end of every complete line

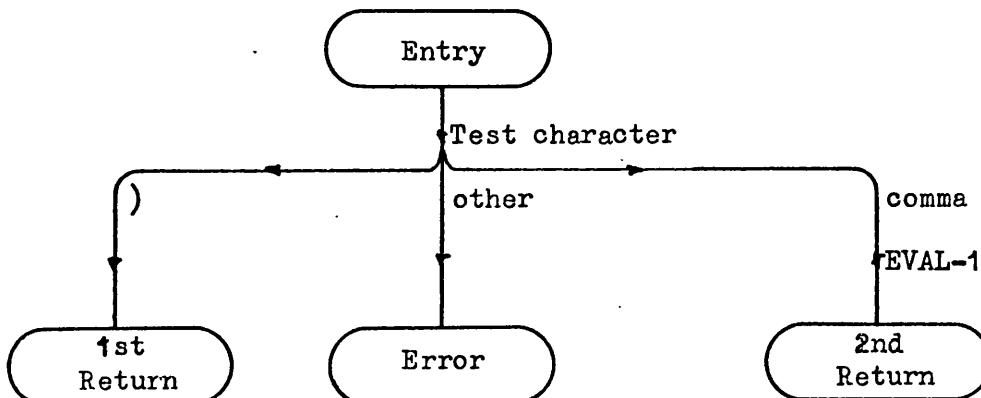


INHIBIT

Entered from a type/ask command if a parameter modification is in progress

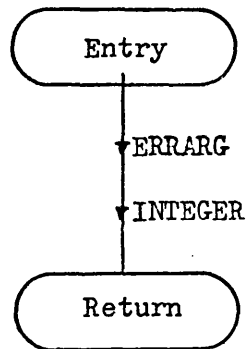
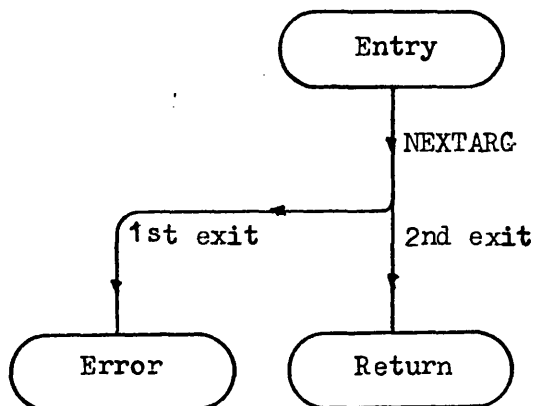
NEXTARG

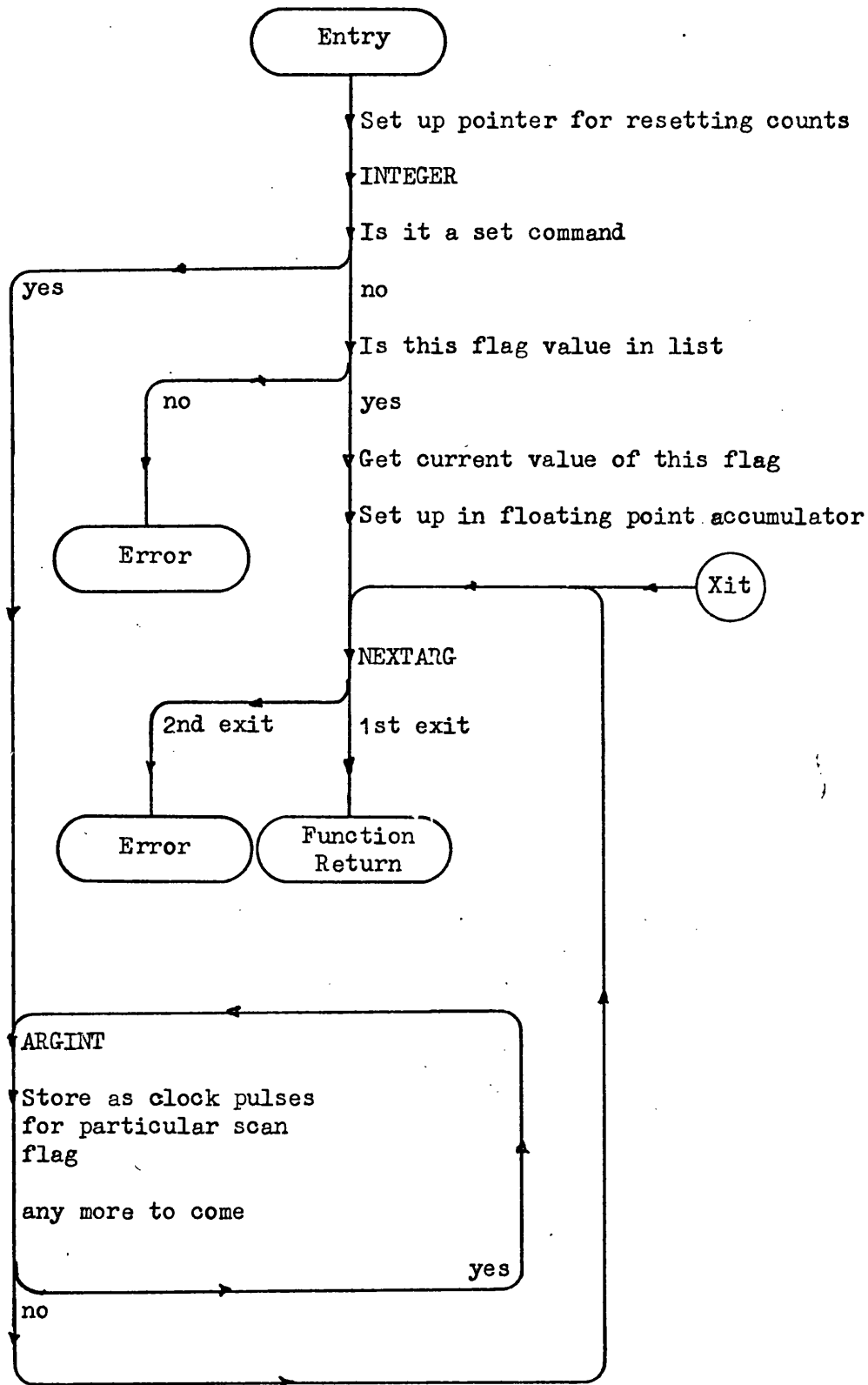
Multiple argument evaluation routine

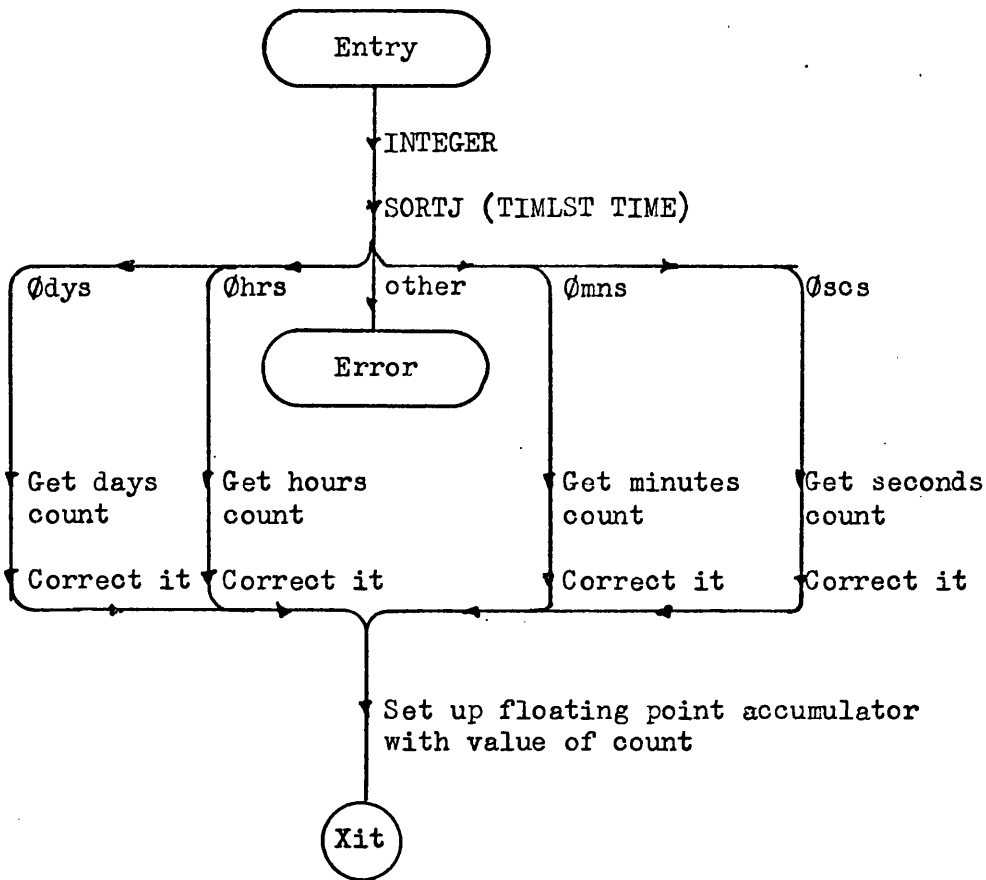
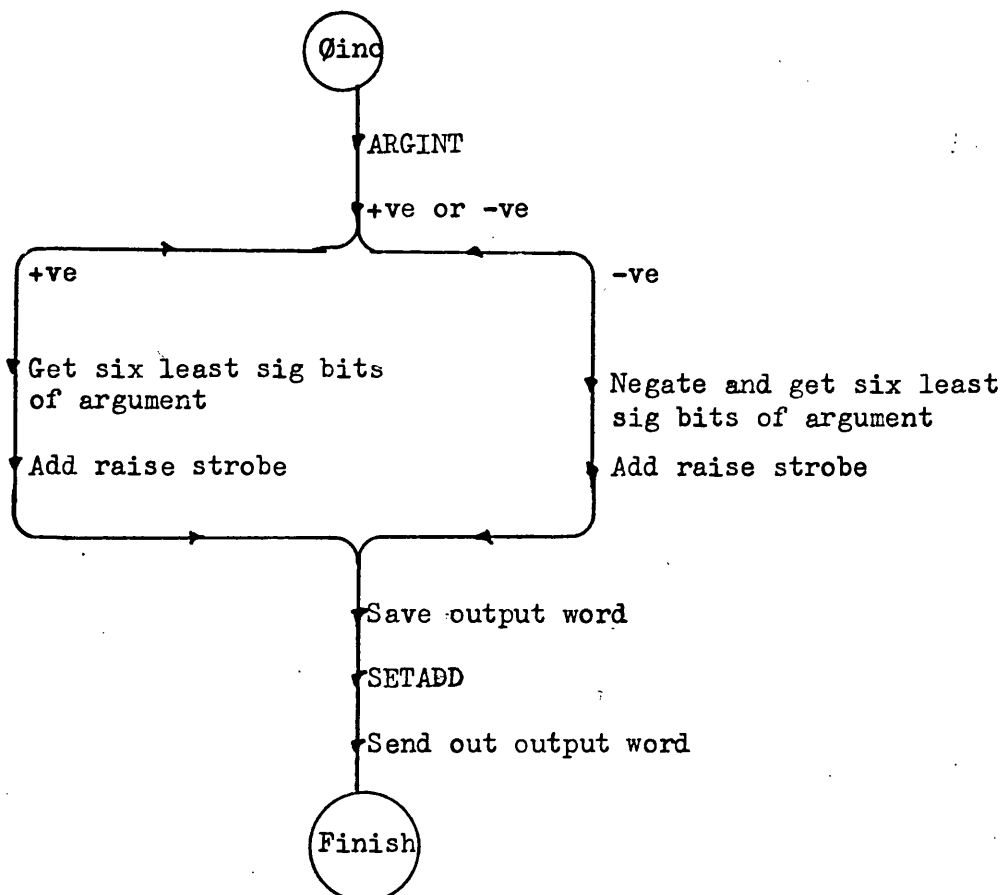


INTARG

Called by ARGINT

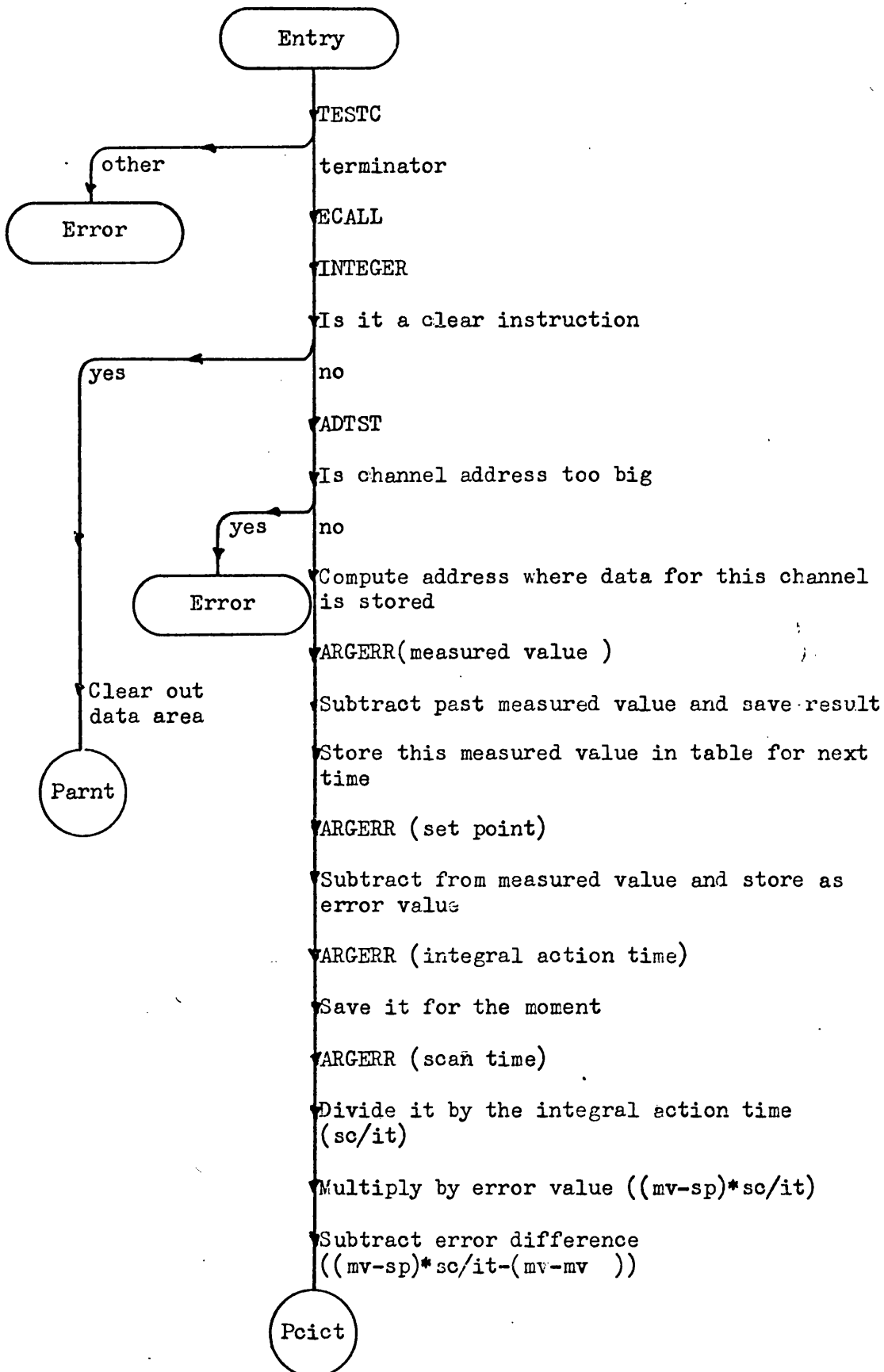
ERRARG

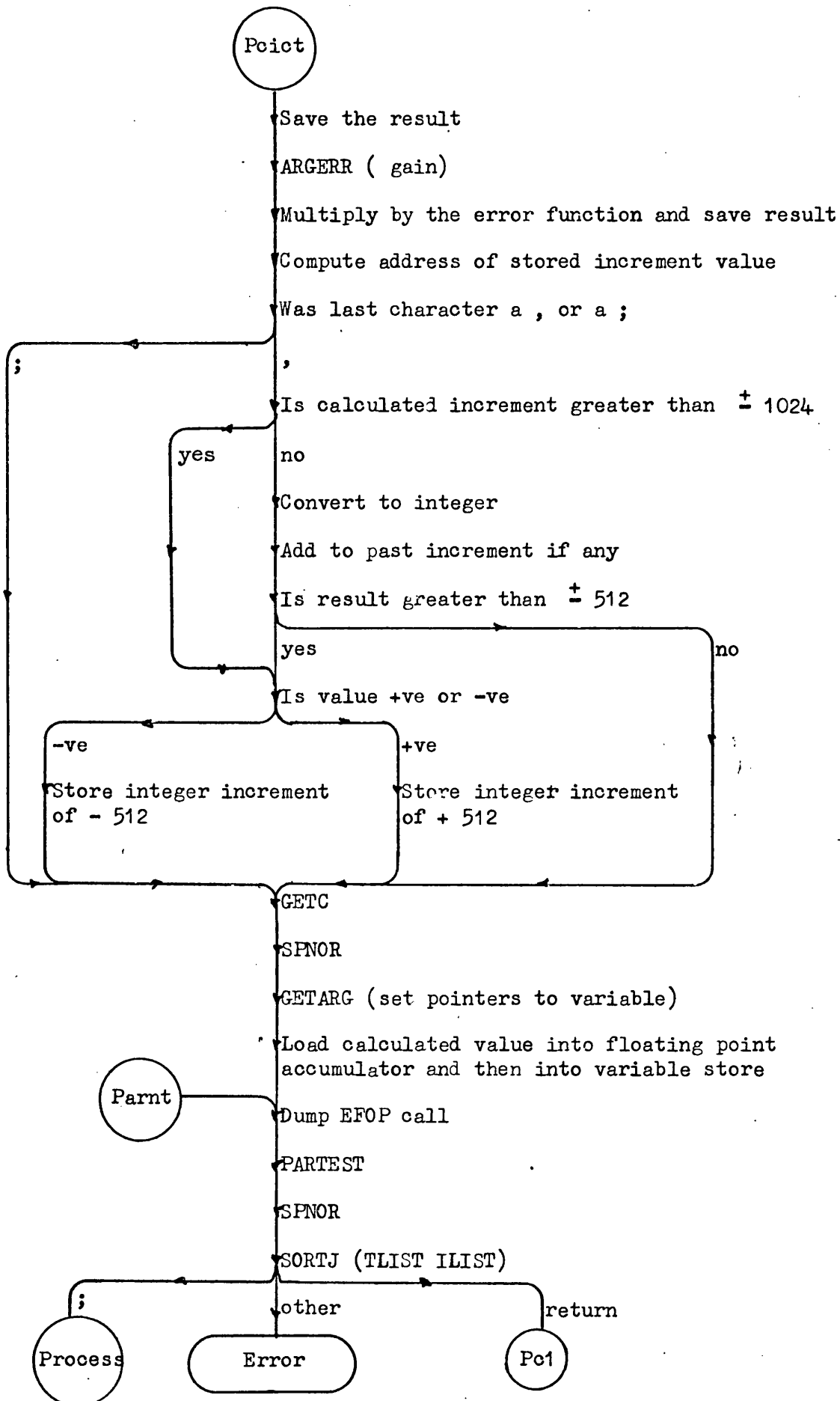
FLAG Routine

FTIM RoutineExtension to FOUT command

PCI New command for PCI control algorithm

Entered from the command decoder



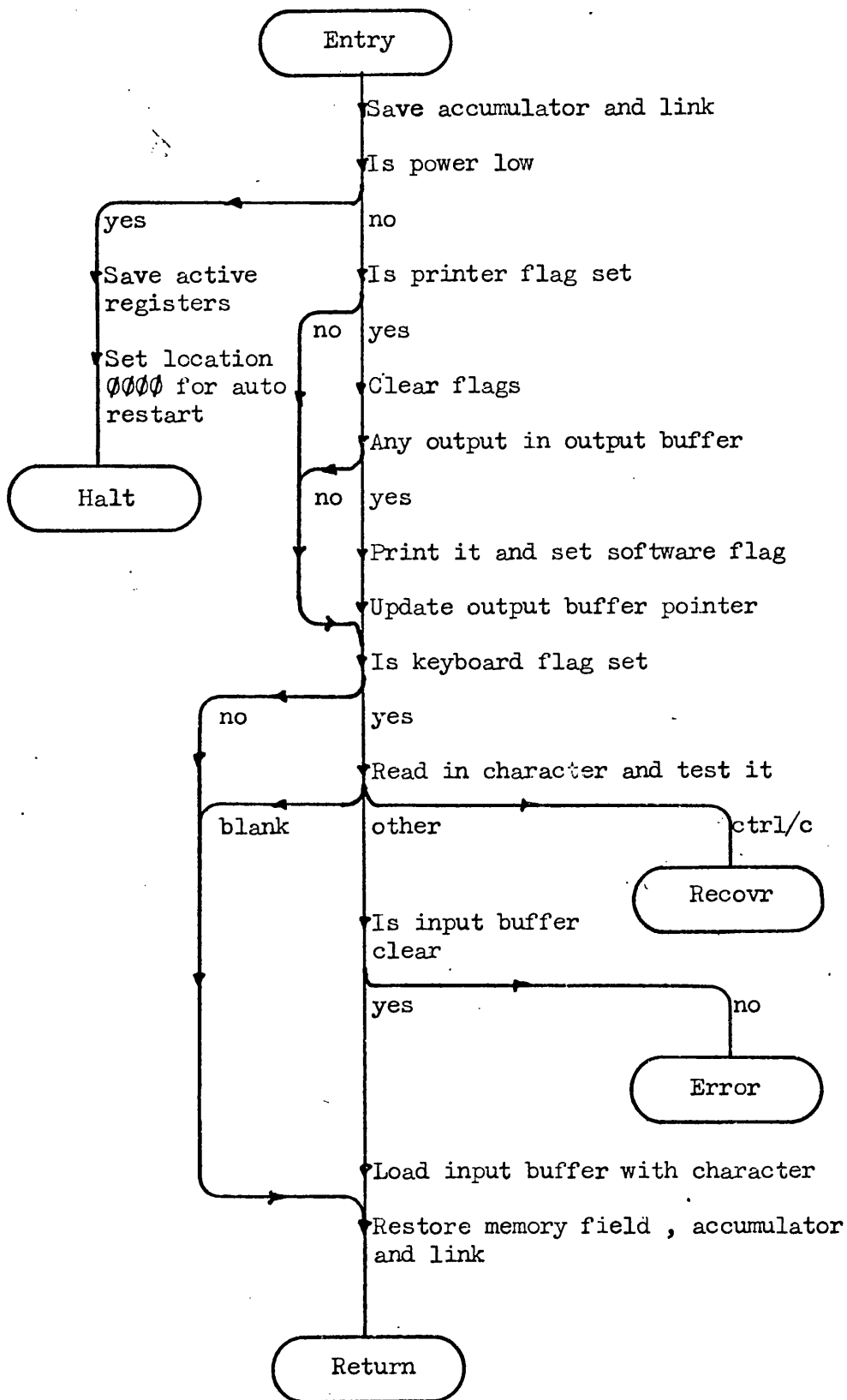


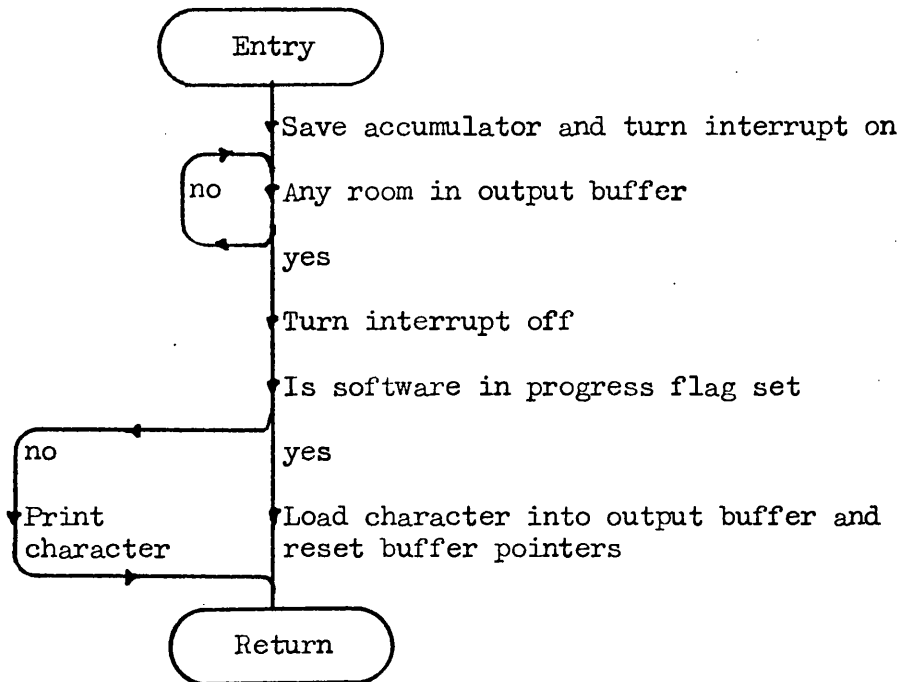
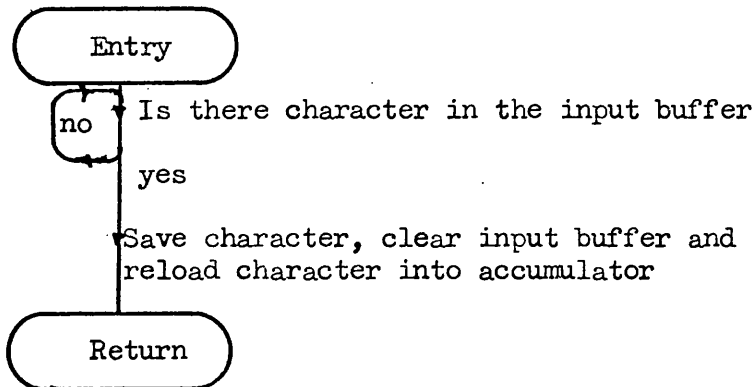
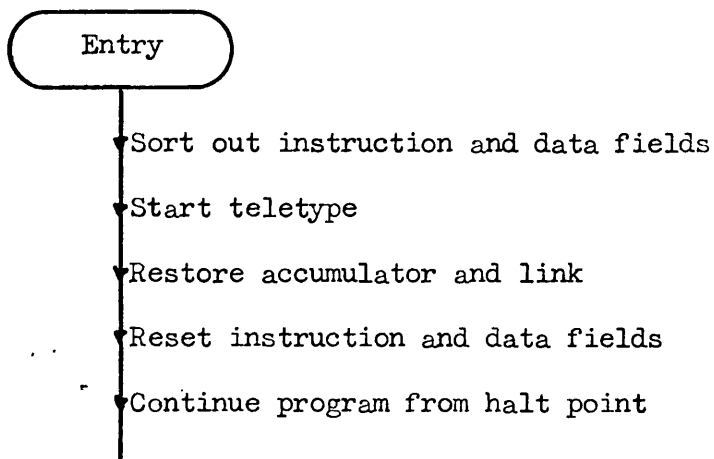
APPENDIX F.

Flow Charts for the File
Monitoring System

Super D Core Map Field 2

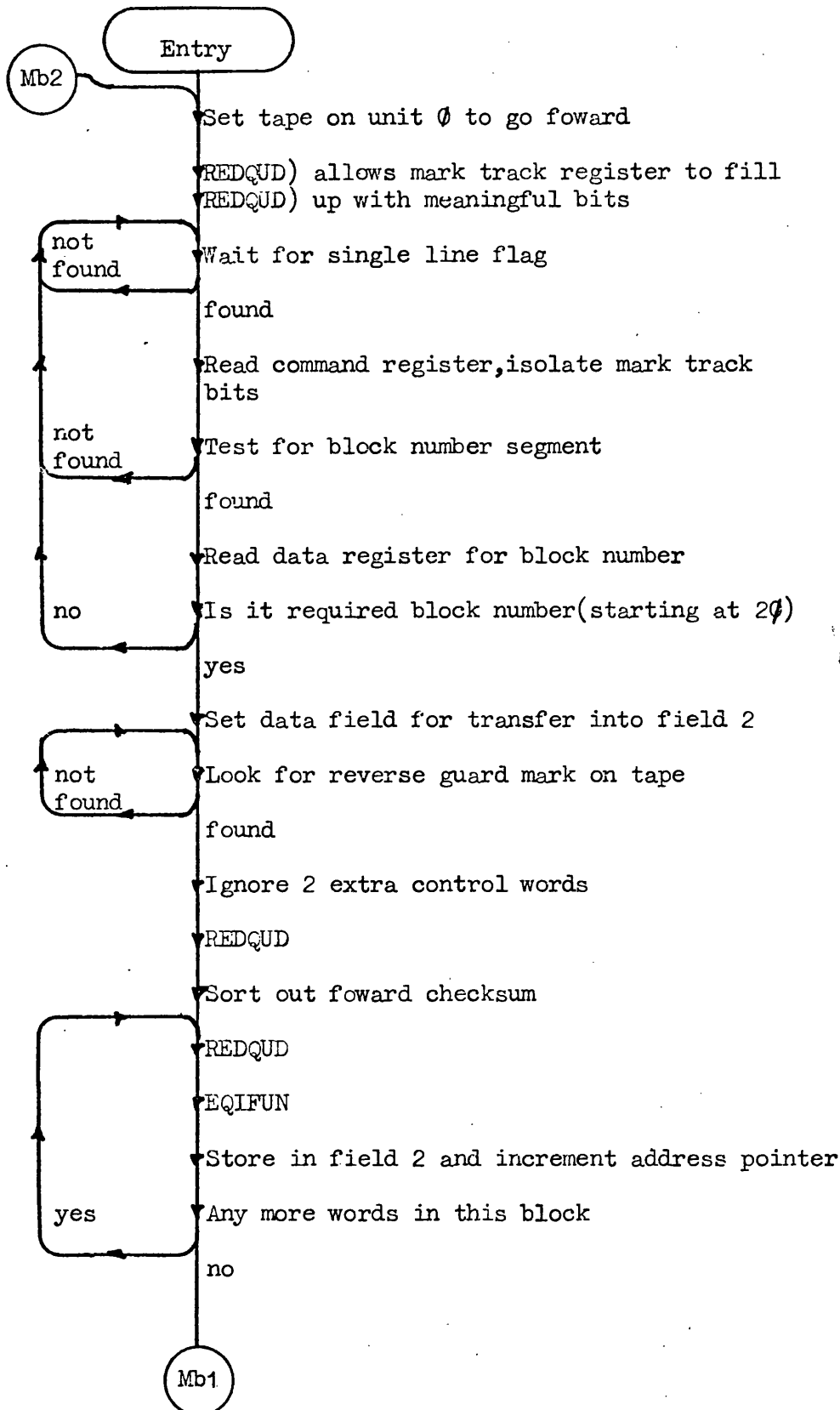
0	Zero Page Pointers													
1	Command Decoder				Directory Read/Write				Free					
2	Interrupt Processor Shifting Routines										Rtl6, Testc and Sortc		Free	
3	Pushdown List Controls				Input		Sortj		Free					
4	Decape Control													
5	Routine													
6	Getc		Readc		Printc		Prntln							
7	Packc		Error Recovery Routine				Free							
10	Rubout		Spnor		Testn		ComLst, Comgo etc				Free			
11	Filstr		Device		Getdvc		Getfil				Find			
12	Flsrch		Directory				Zero							
13	Save													
14	Erase				Inouts		Seto		Inout					
15	Copy										Octornt			
16	Run		Sa		Testor		Filedvc		Free					
17	Load		Save				Free							
20	Help													
21	Write and Access Commands													
22	Free													
23	Free		Command Input Buffer											
24														
25														
26														
27	5000 to 7177													
30	Directory Buffer													
31	Also used as a Read/Write Buffer													
32	For Copy and Erase													
33														
34														
35														
36	Free													
37	Free													

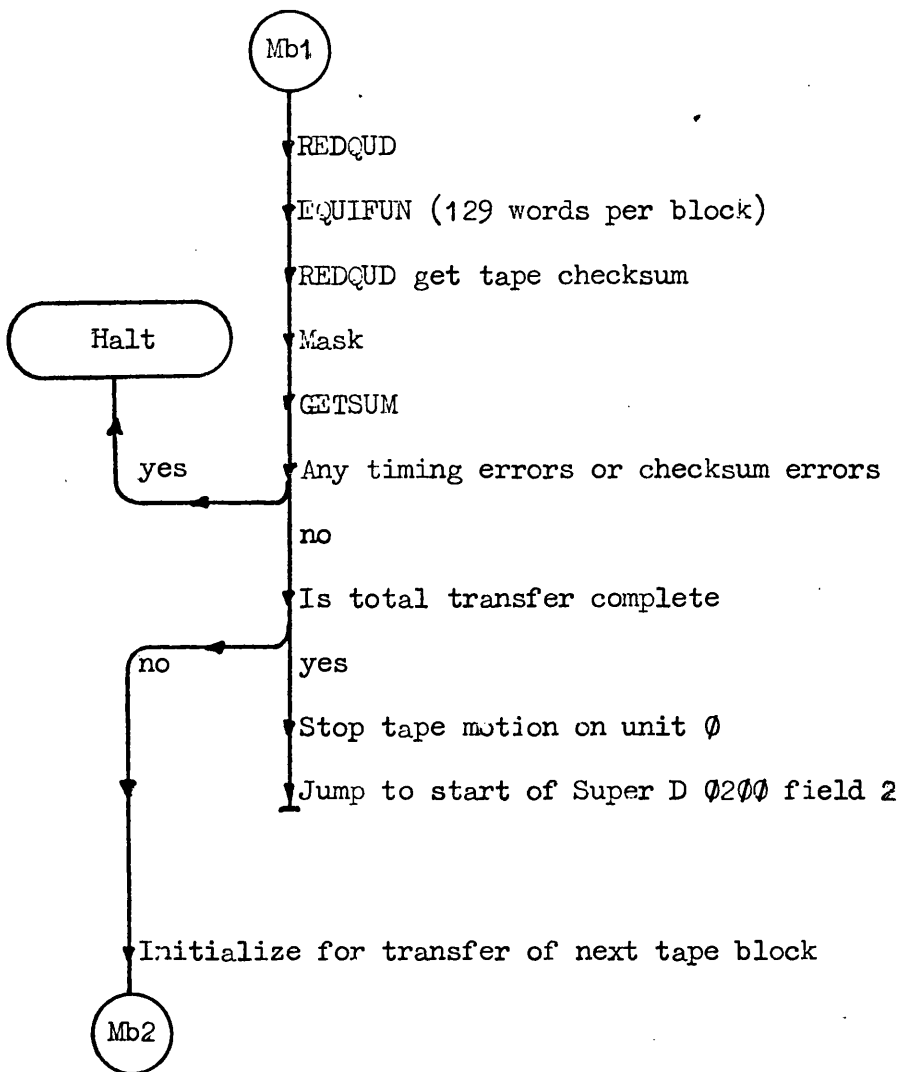
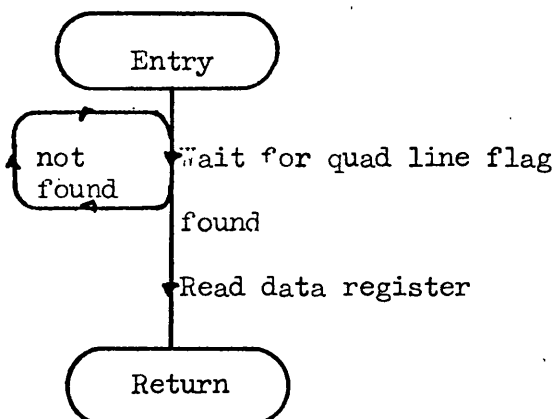
Super D Interrupt Processor

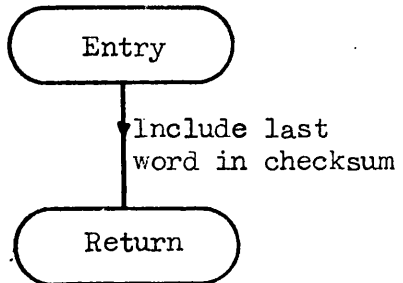
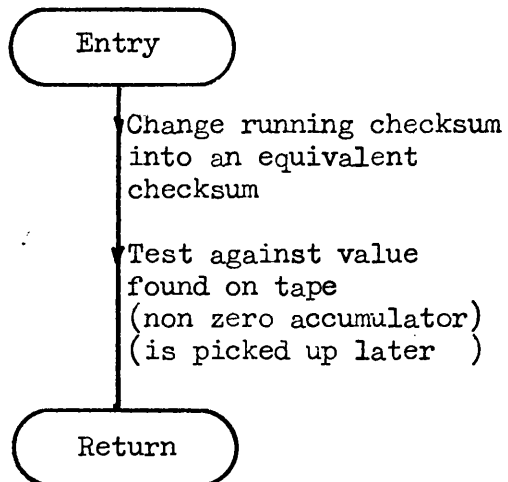
Teletype Output RoutineTeletype Input RoutinePower on Automatic Restart Routine

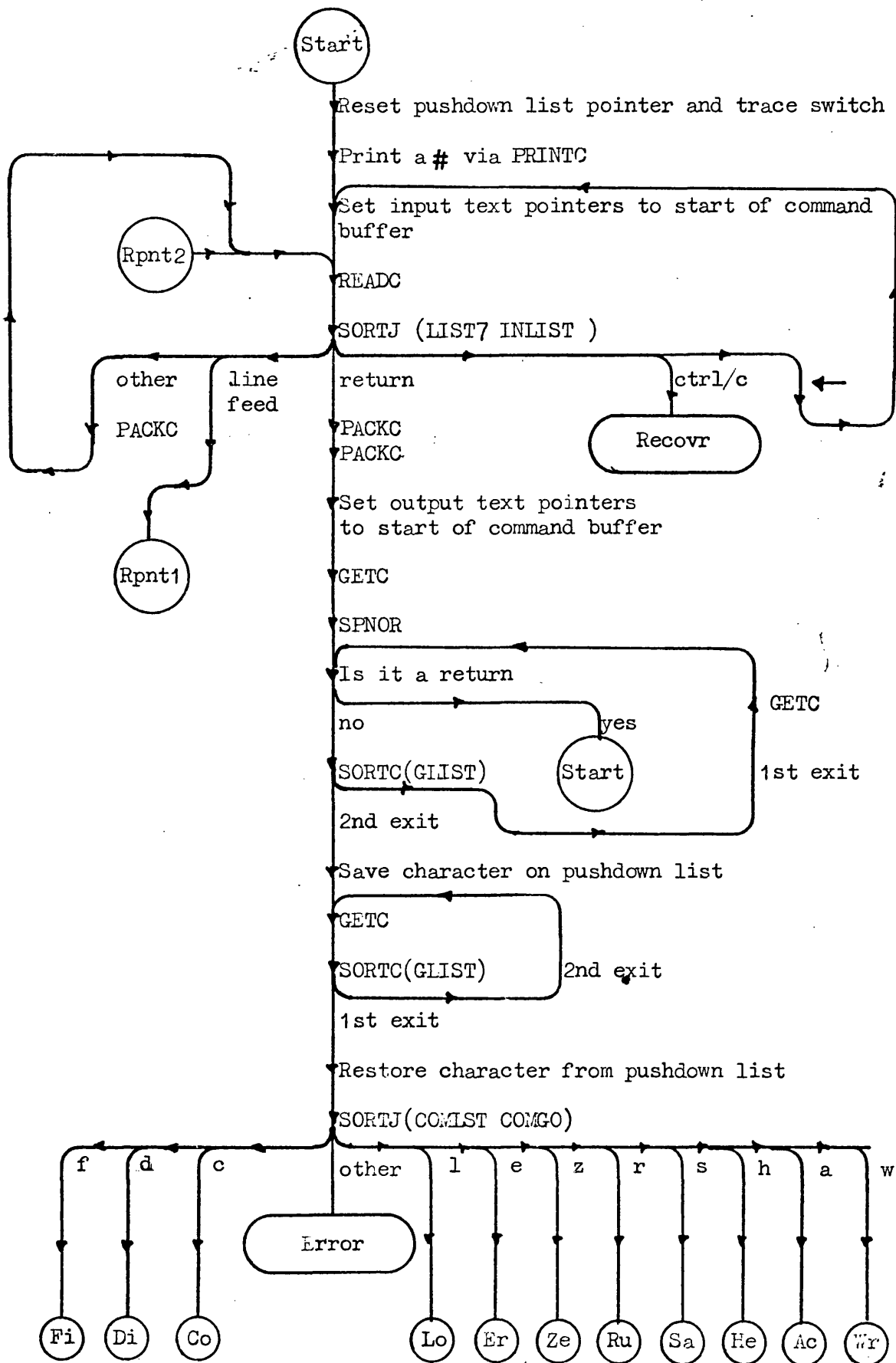
Super D Maxi Bootstrap

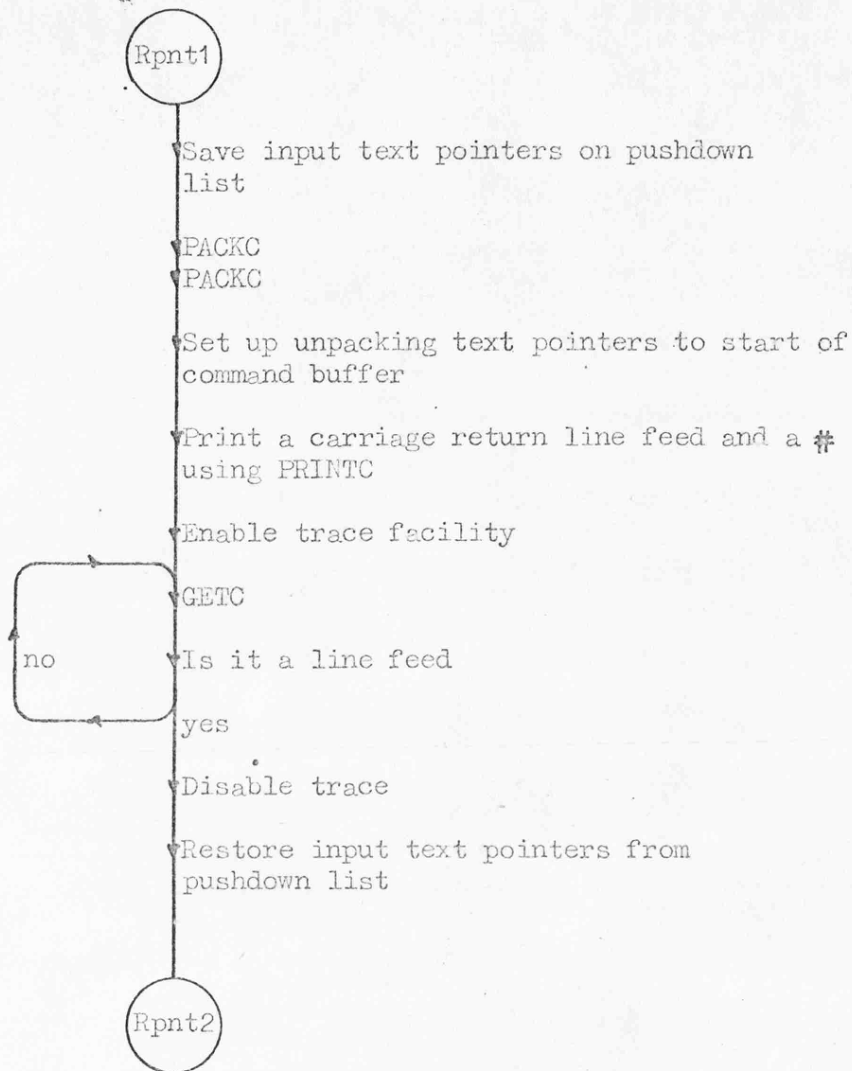
Entered from the Mini Bootstrap



REDQUD

EQIFUNGETSUM

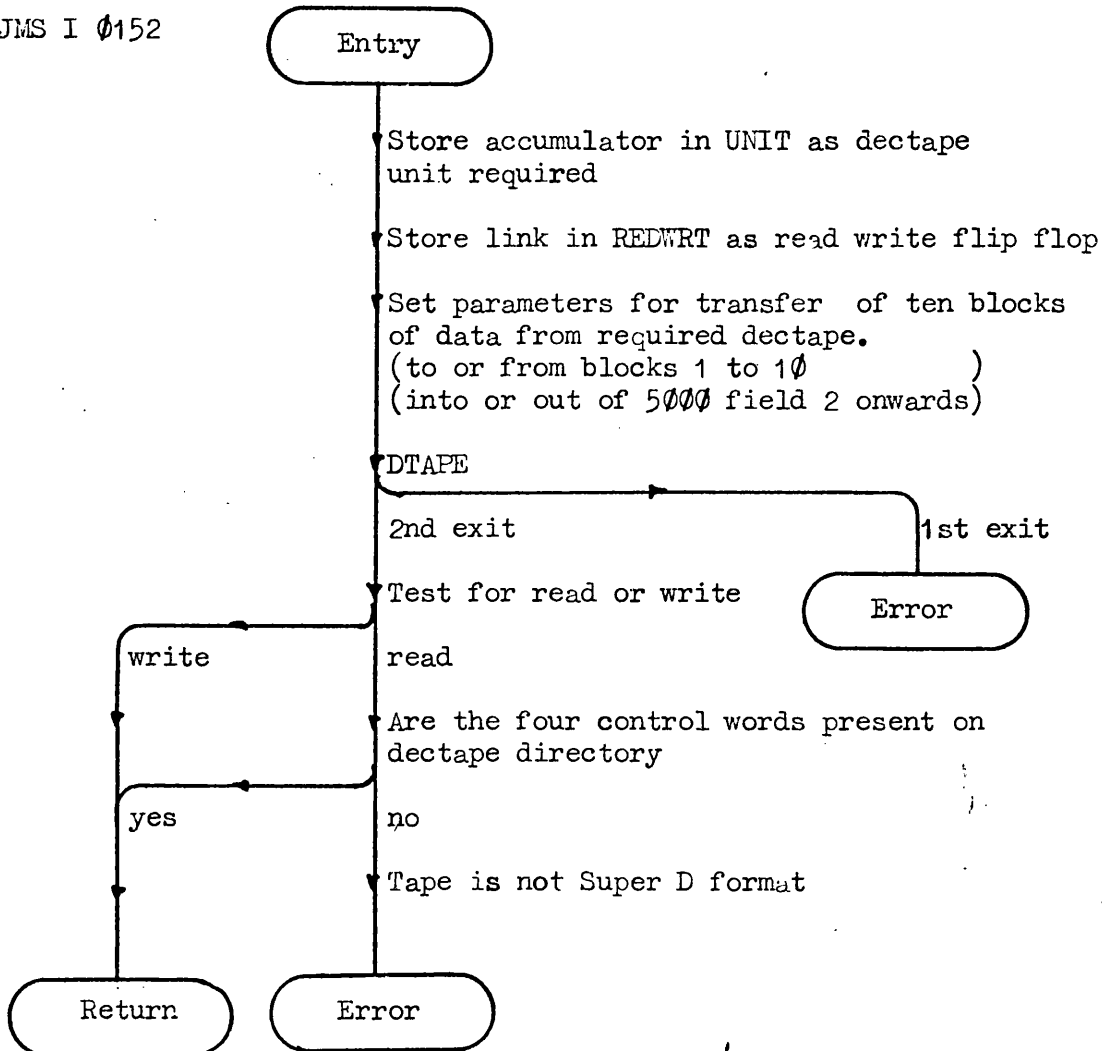
Super D Command Decoder



Directory Read/Write Routine

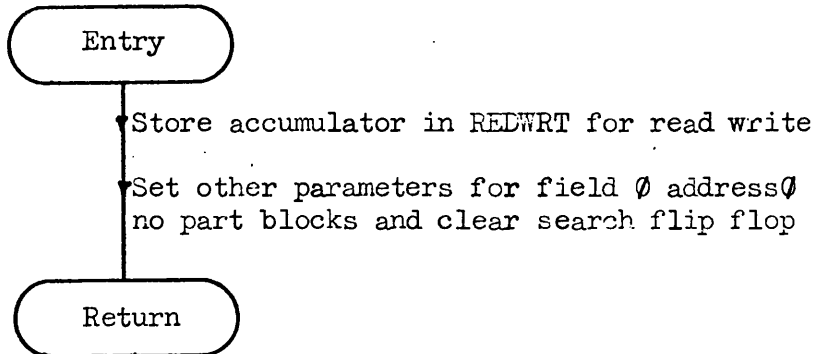
Called by DRCTRY

JMS I 0152

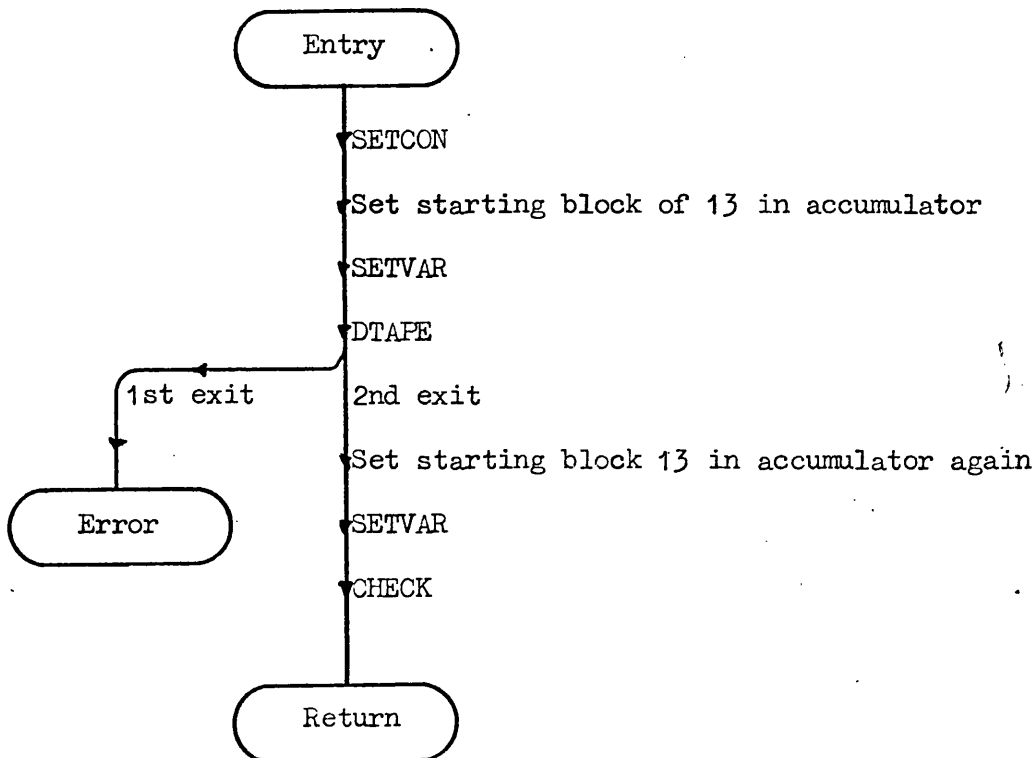


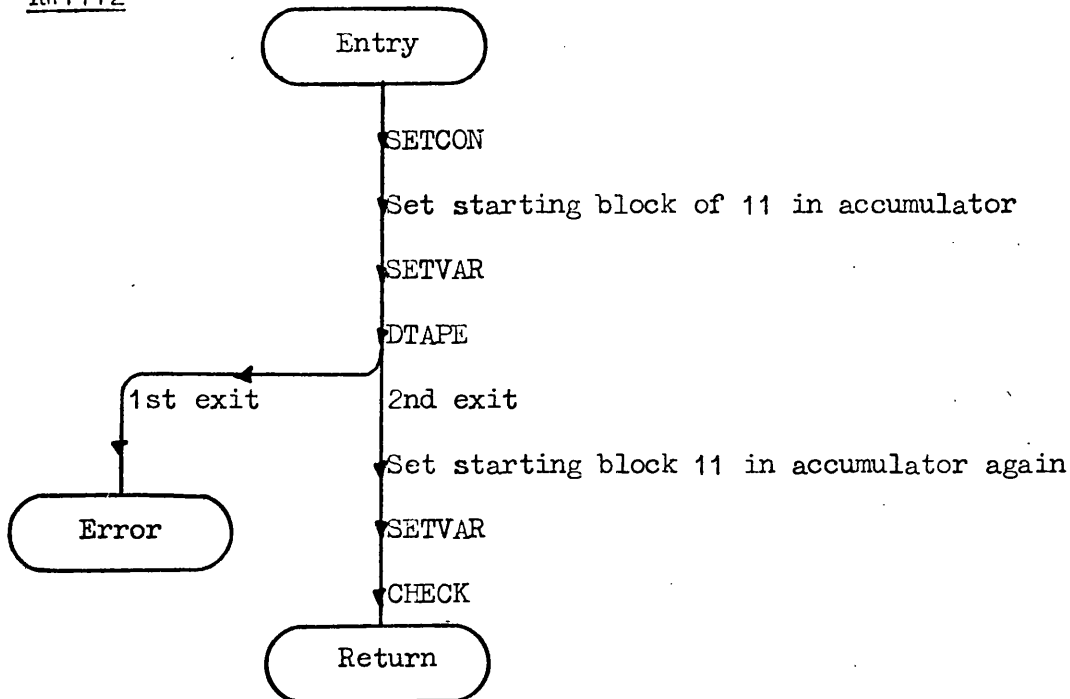
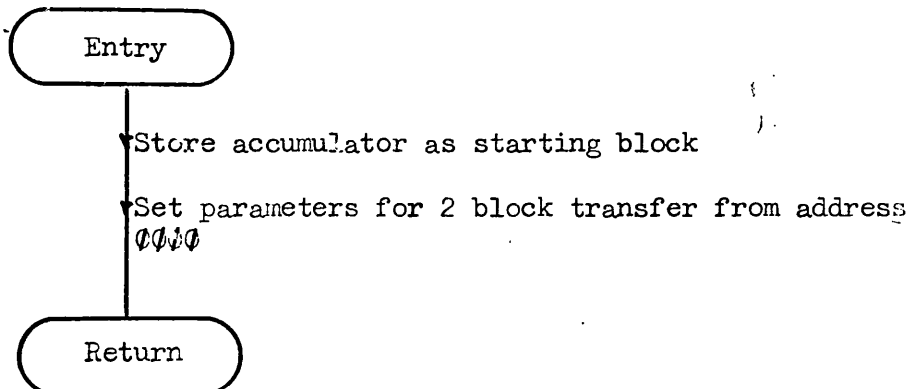
Routines for setting up and saving the interrupt processor

SETCON

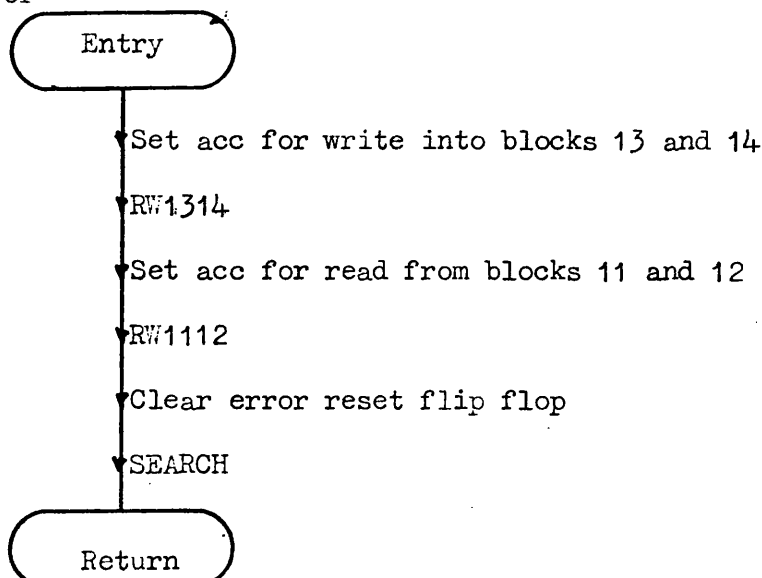


RW1314



RW1112SETVARRSTRT

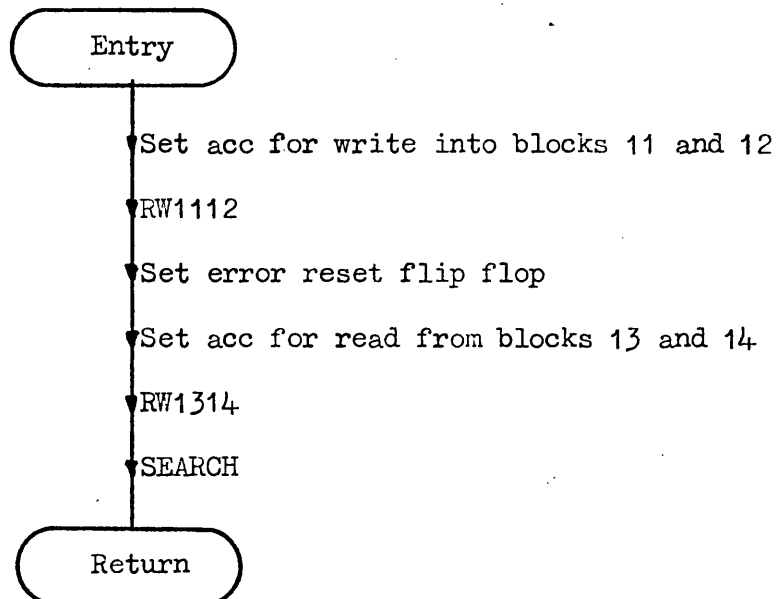
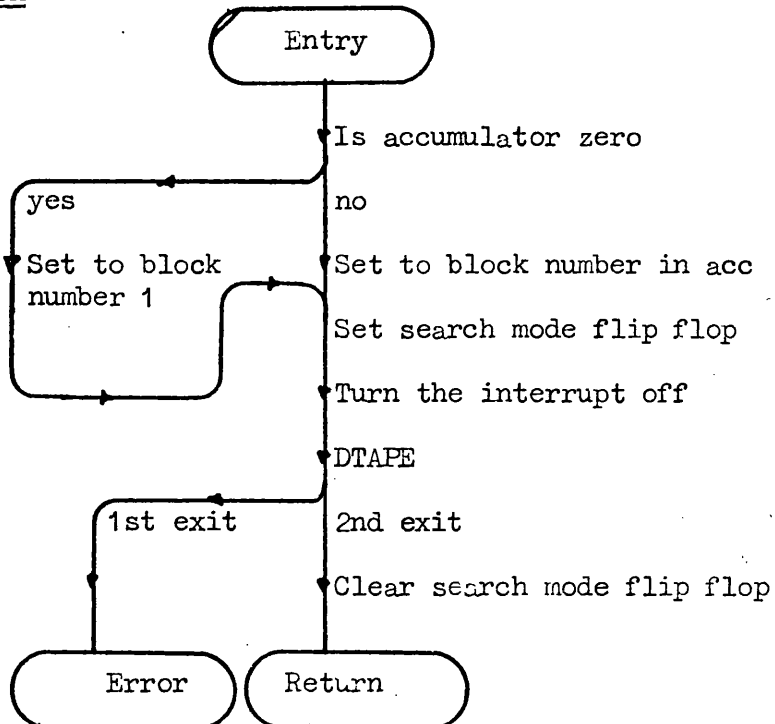
Called by RESWOP
JMS I 0150



TERMN8

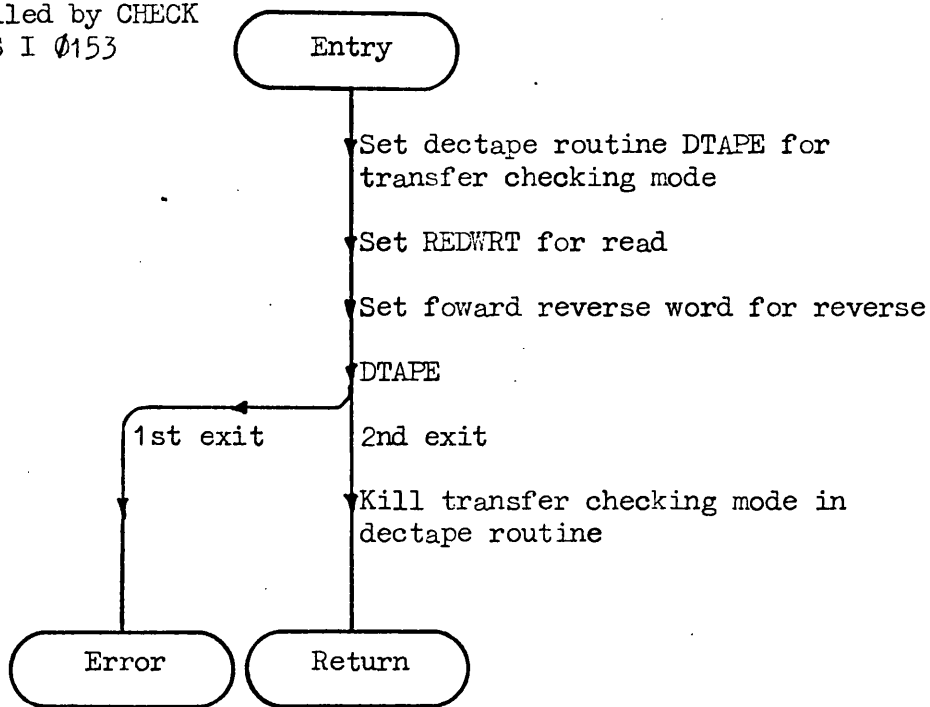
Called by XTRMN8

JMS I 0146

SEARCH

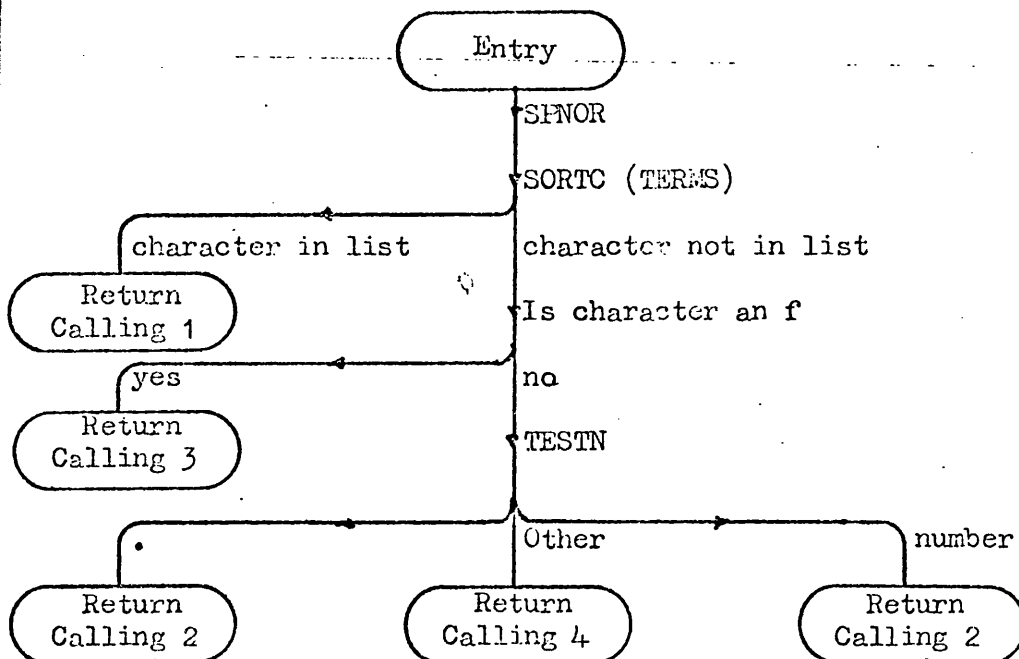
TCHECK

Called by CHECK
JMS I 0153



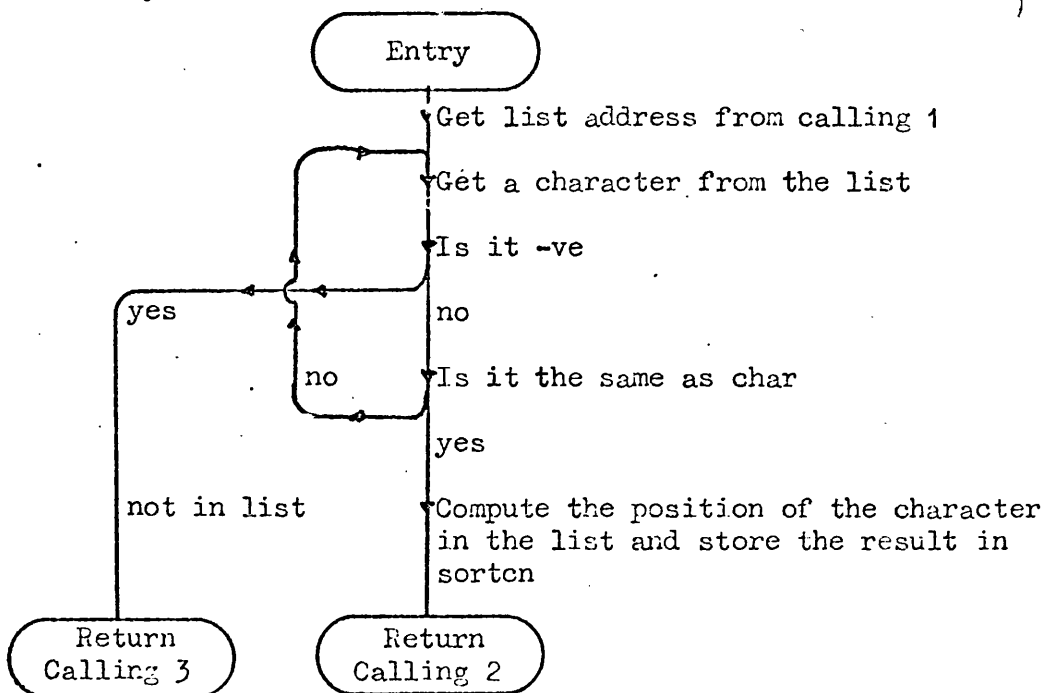
XTESTC

Called by TESTC



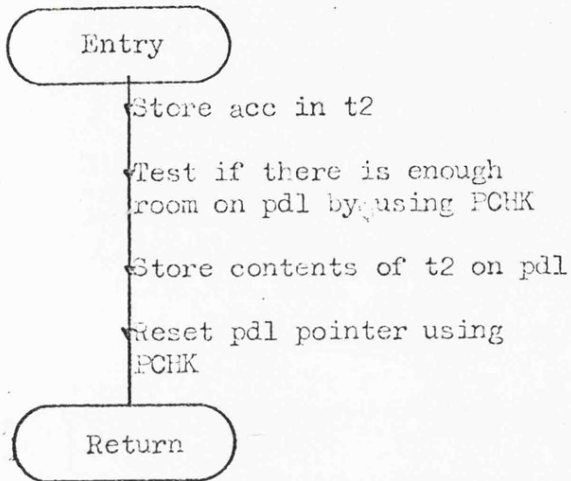
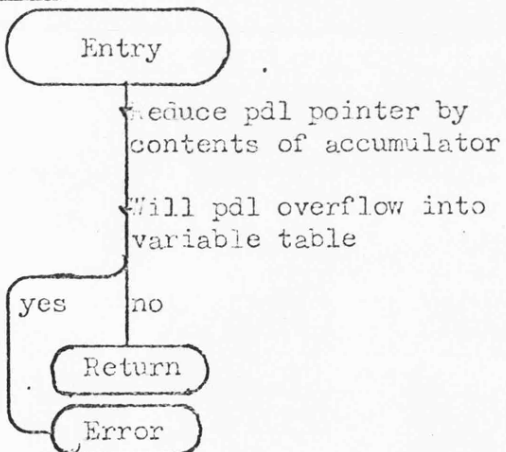
XSORTC

Called by SORTC

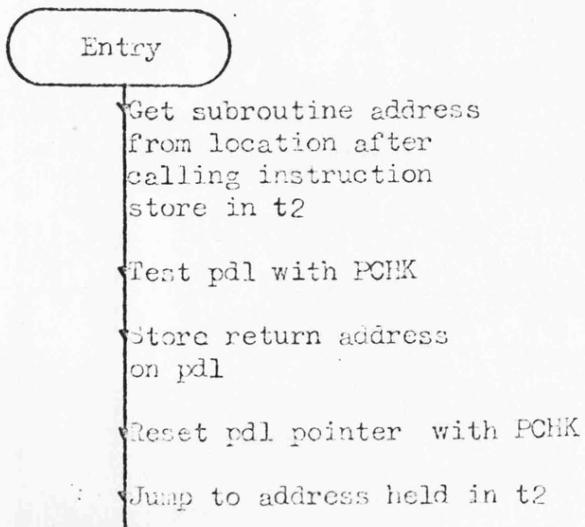


Pushdown List ControlsXIUSHA

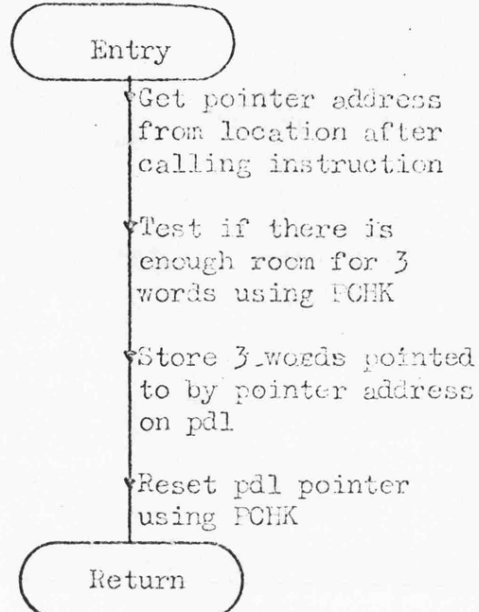
Called by PUSHX

PCHKXPUSHJ

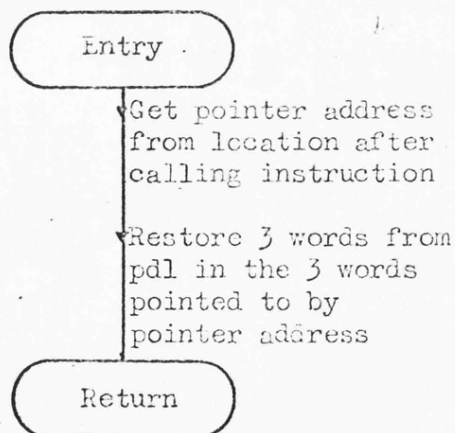
Called by PUSHJ

PD2

Called by PUSHF

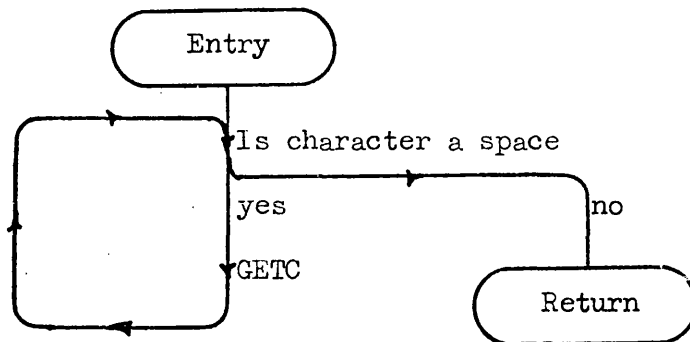
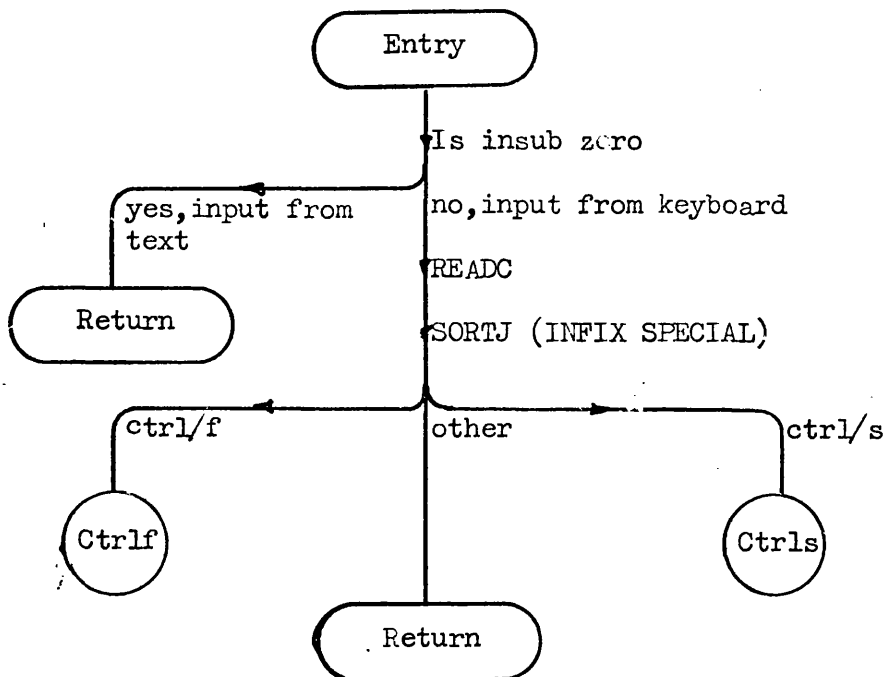
PD3

Called by POPF



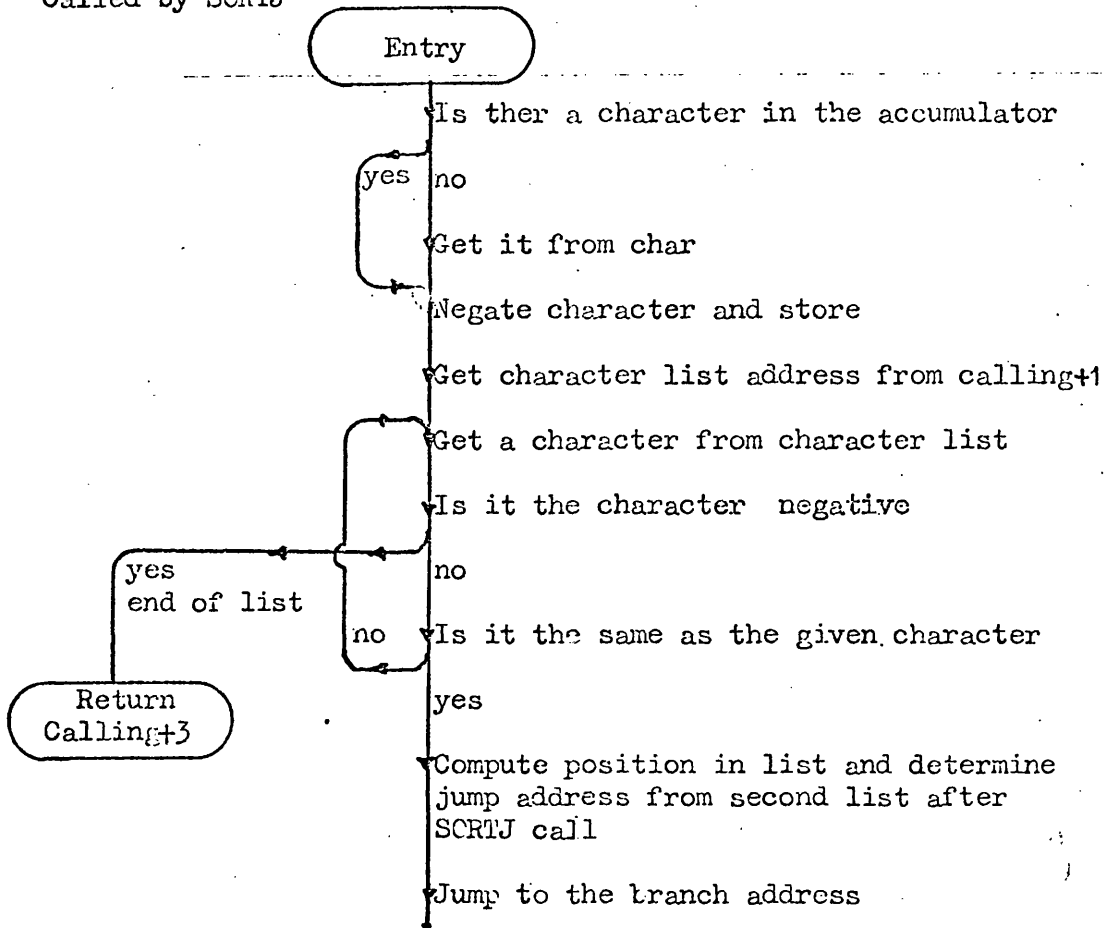
XSPNOR

Called by SPNOR

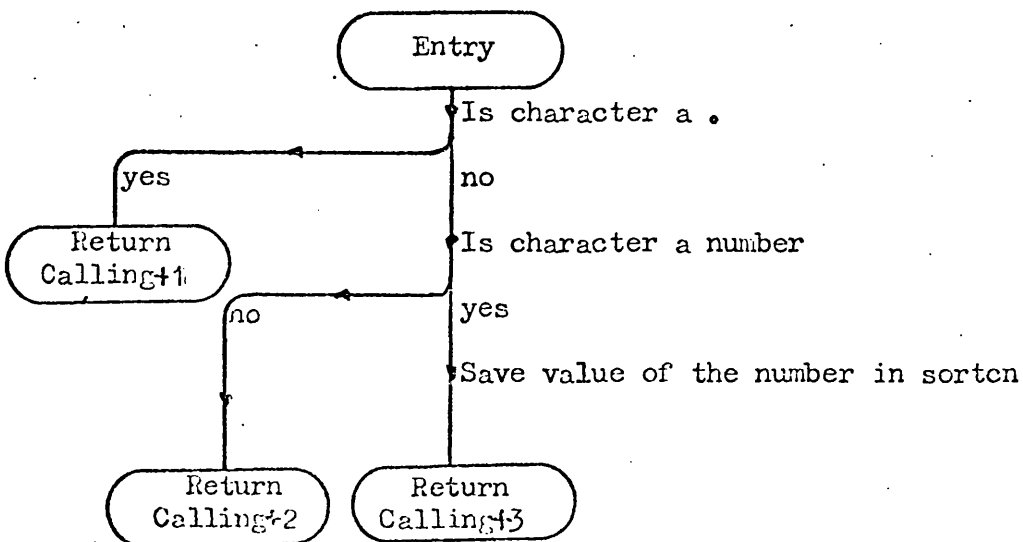
INPUT

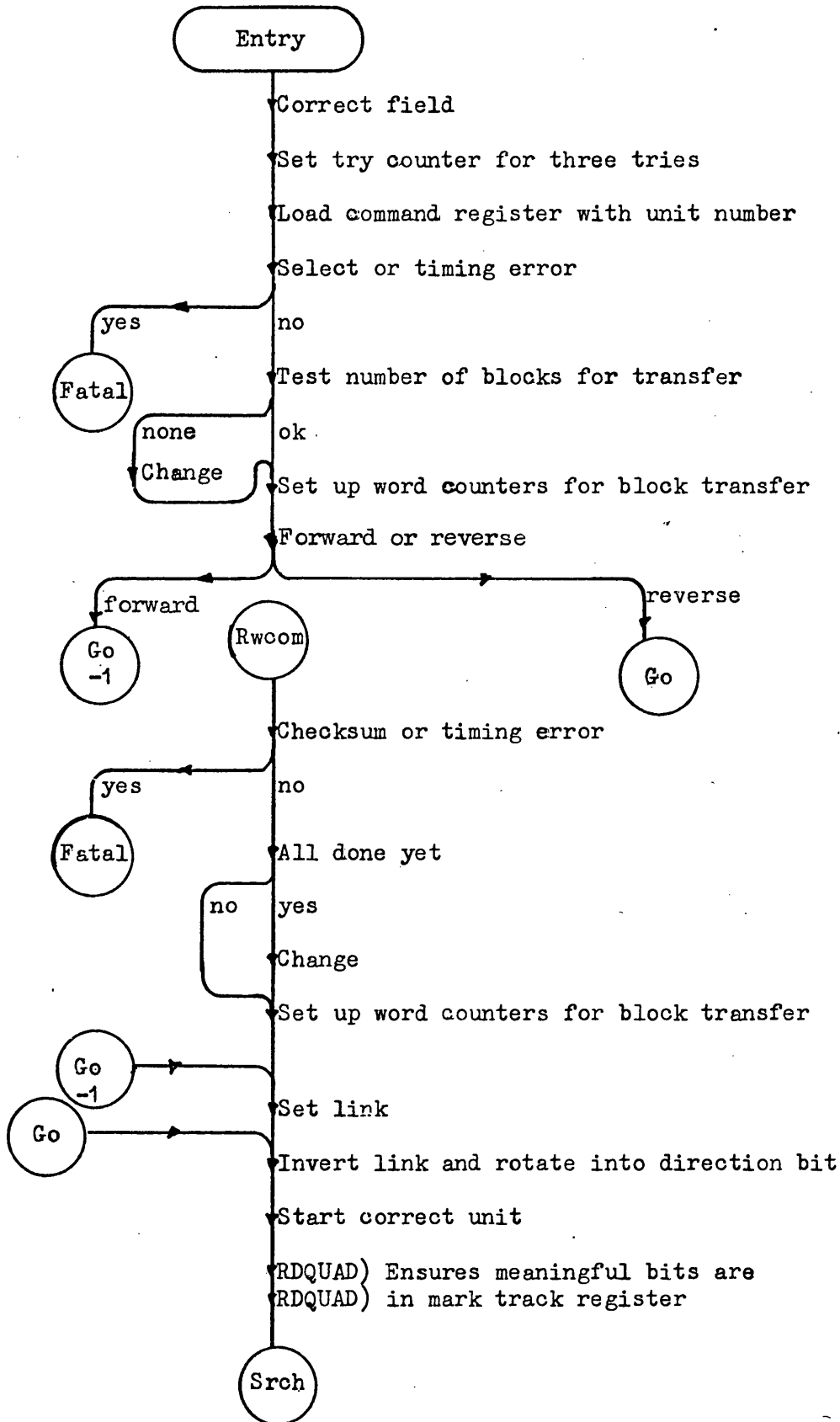
SORTB . Sort and Branch Routine

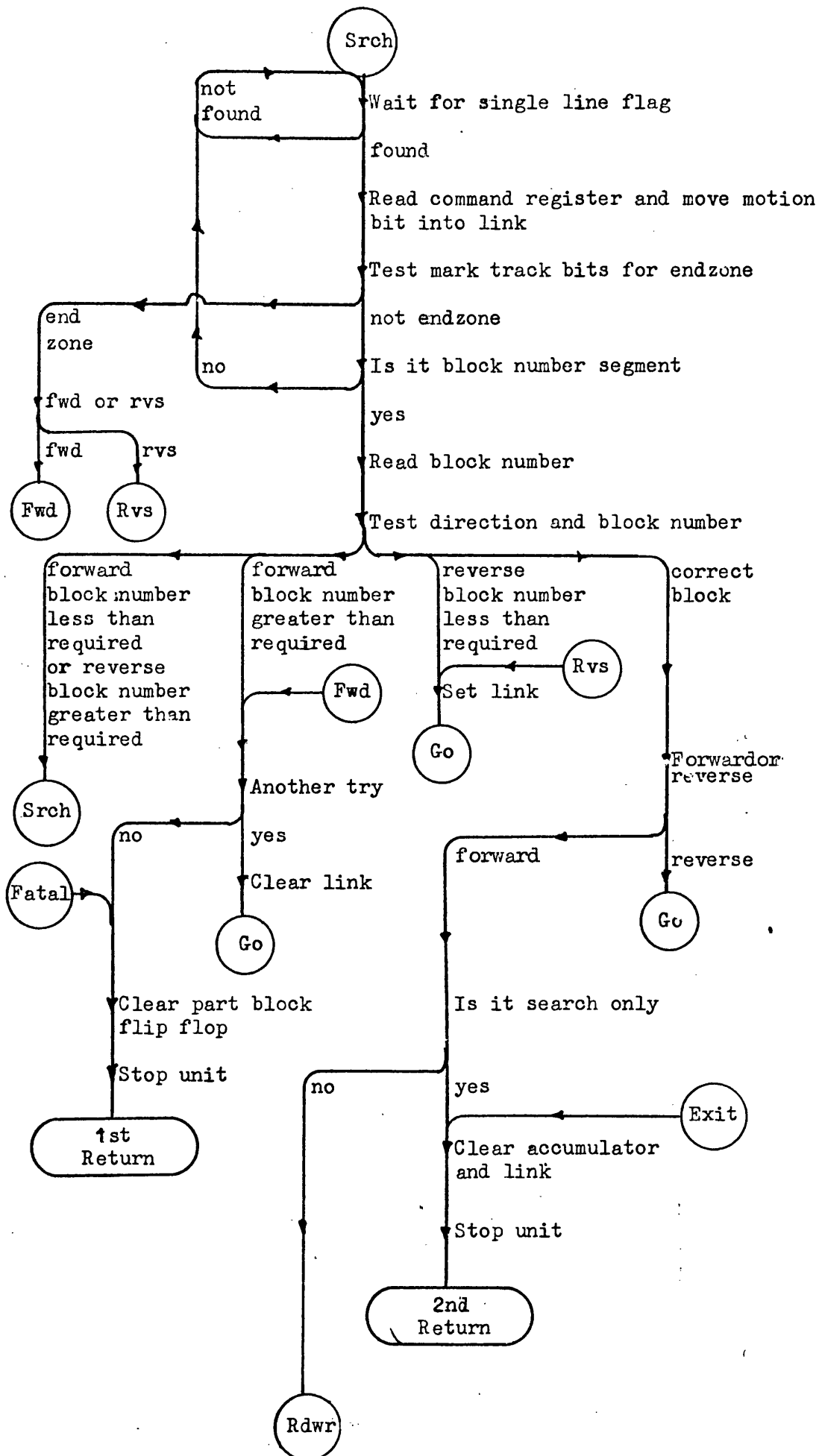
Called by SORTJ

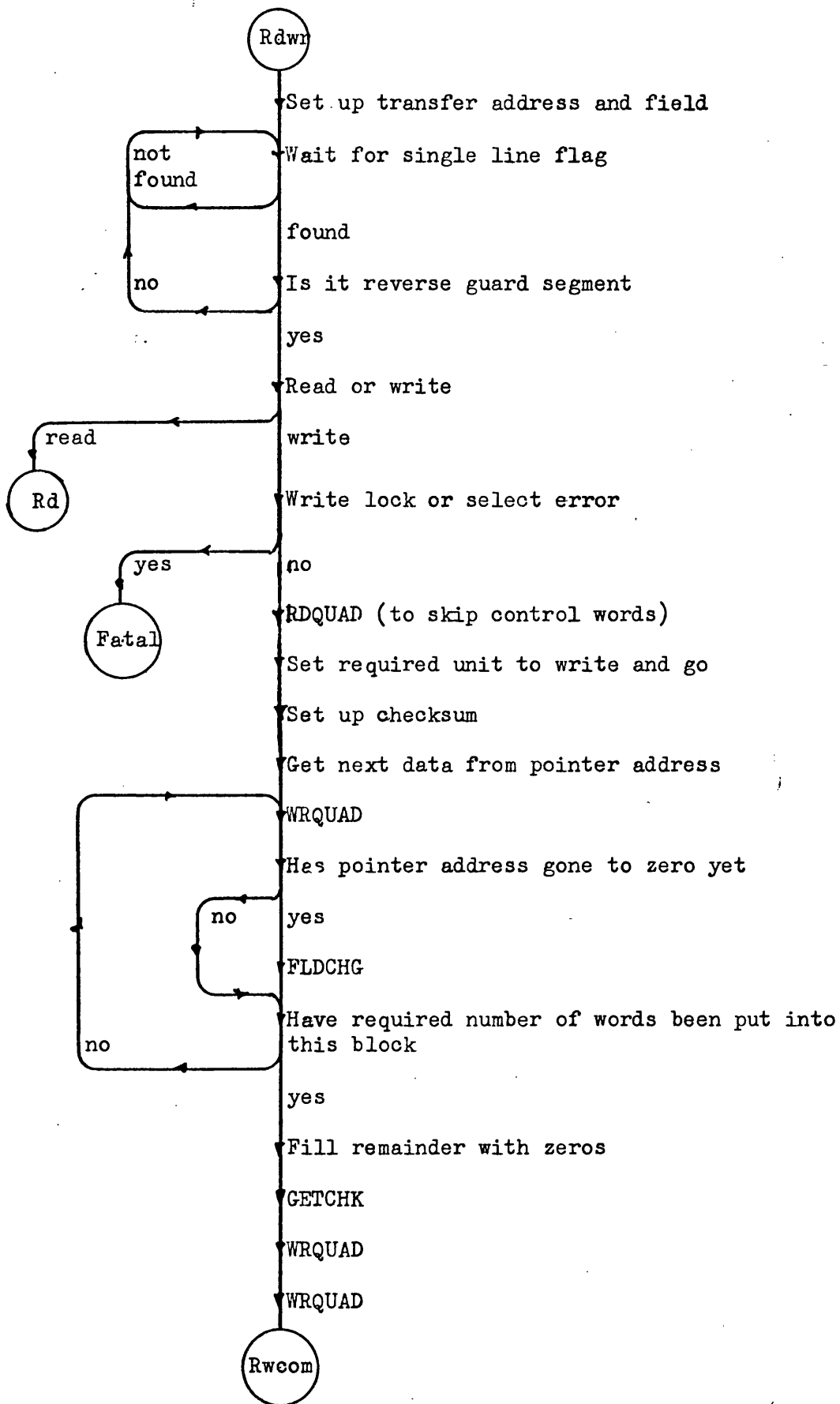
XTESTN.

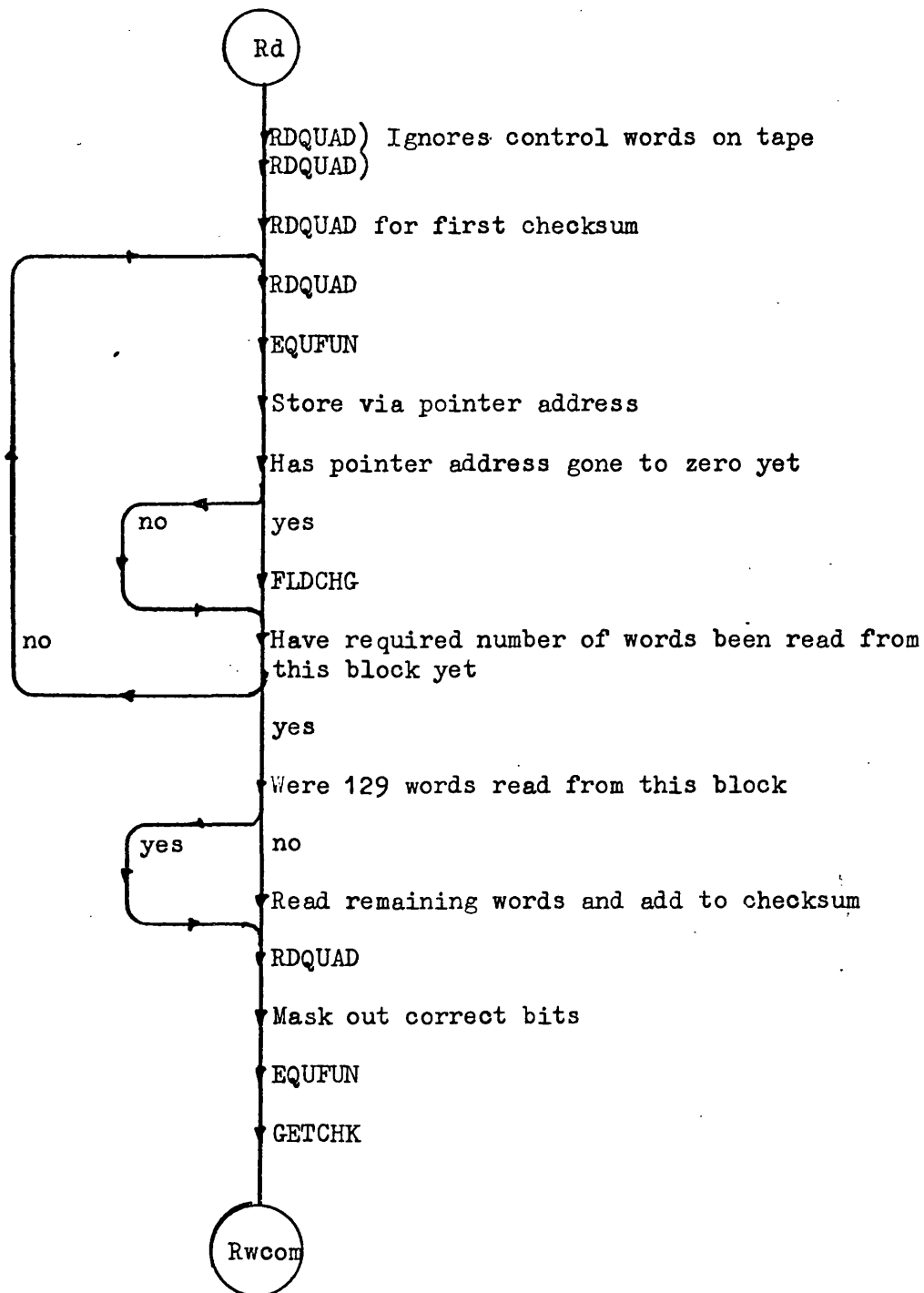
Called by TESTN

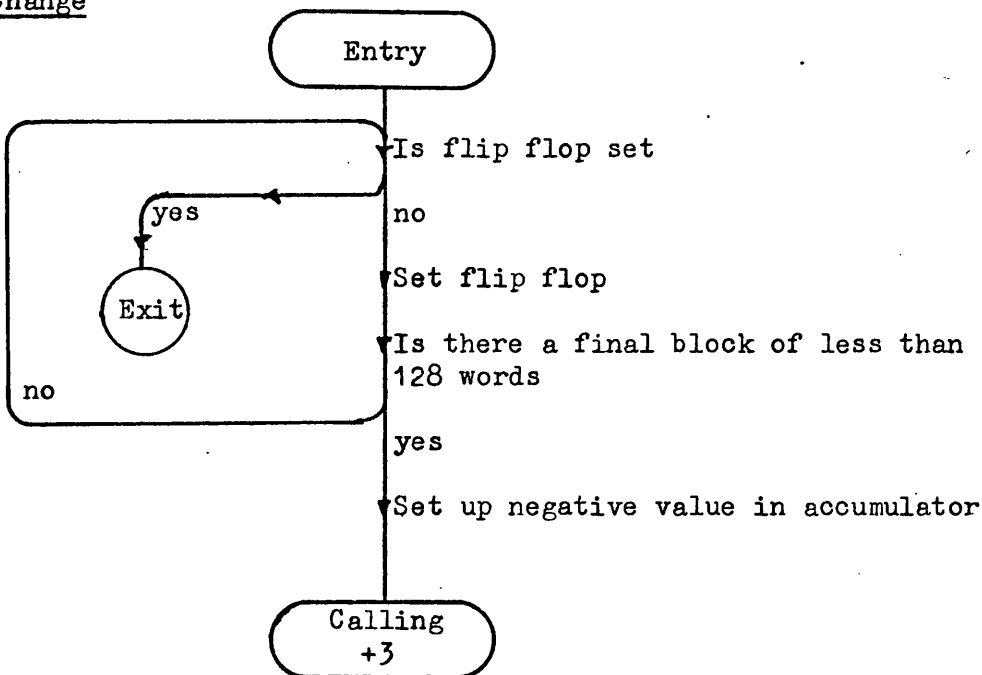
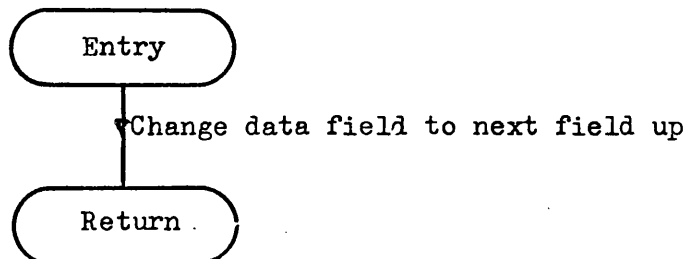
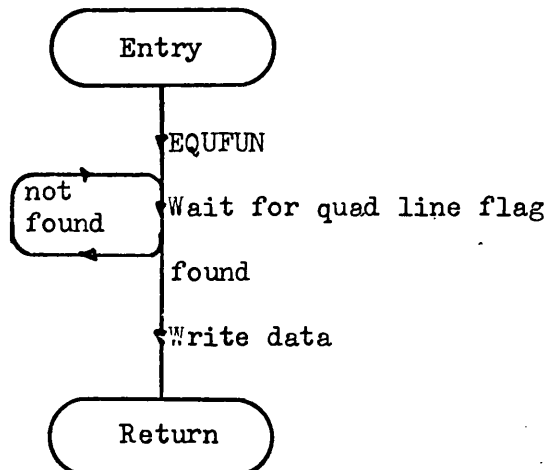
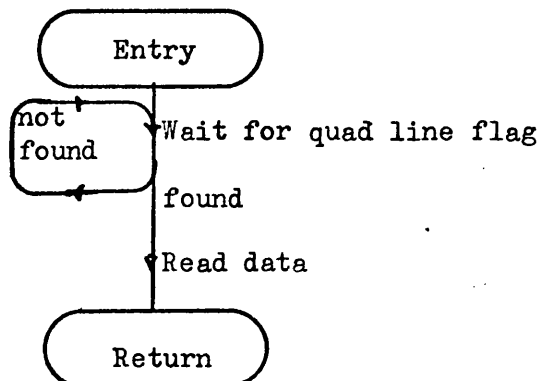


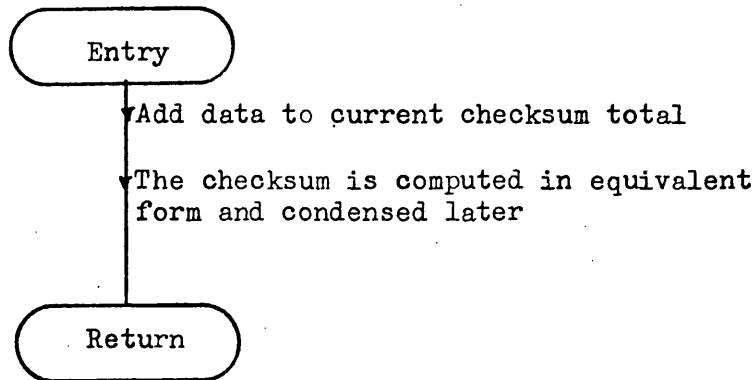
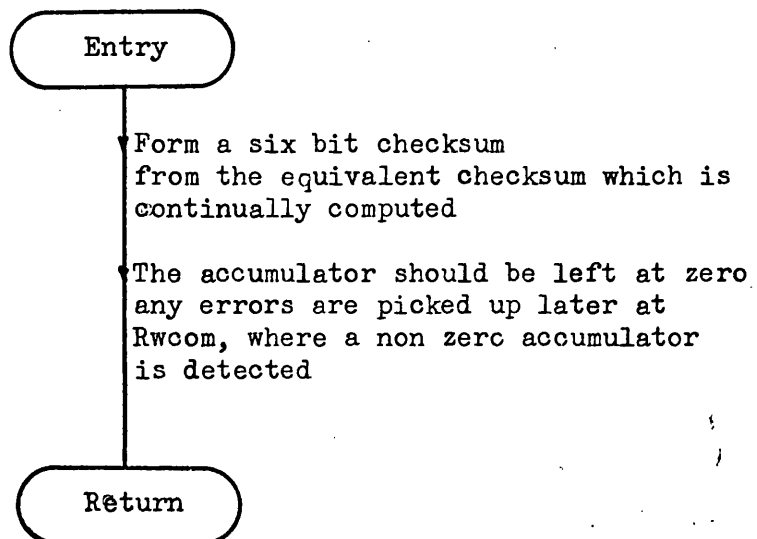
DTAPE . Dectape Handler for Super D



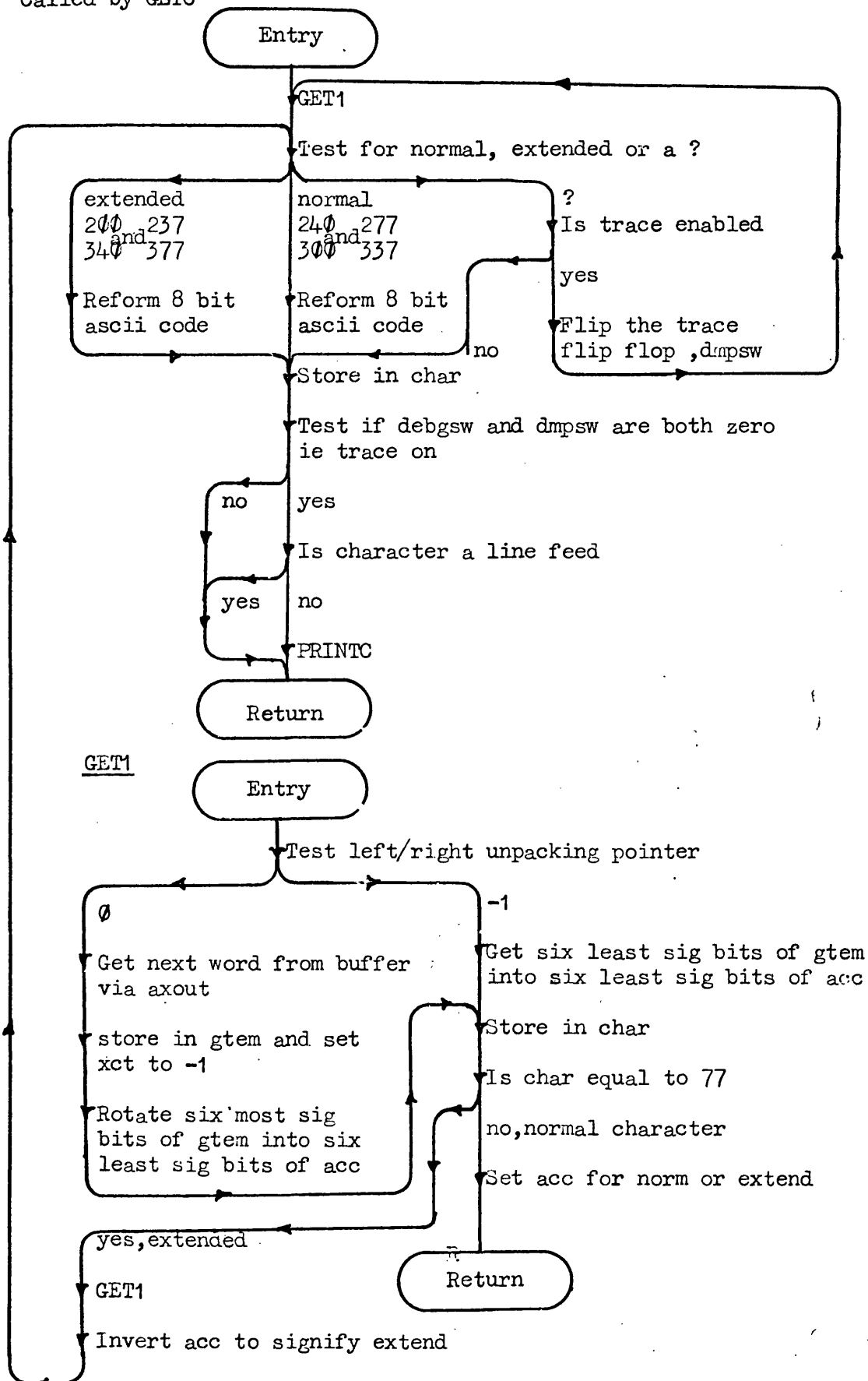




ChangeFLDCHGWRQUADRDQUAD

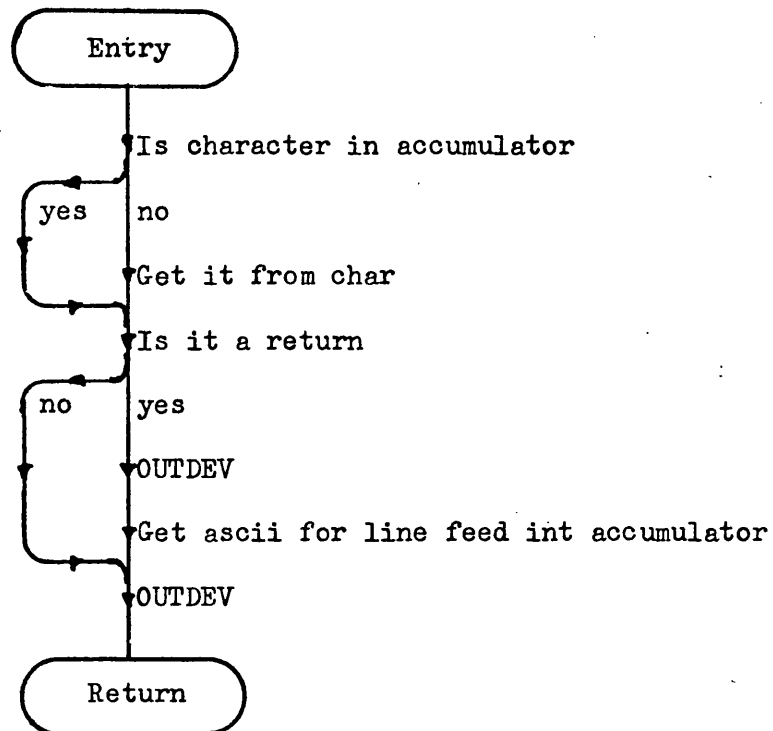
EQUFUNGETCHK

UTRA. Unpack and Reform a Character from Buffer
 Called by GETC

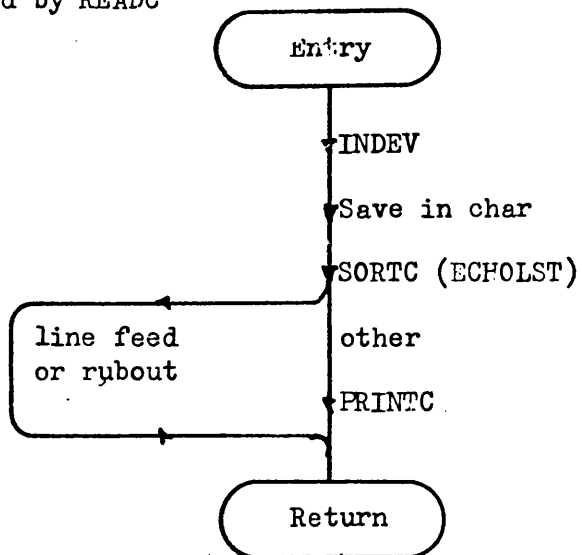


Out

Called by PRINTC

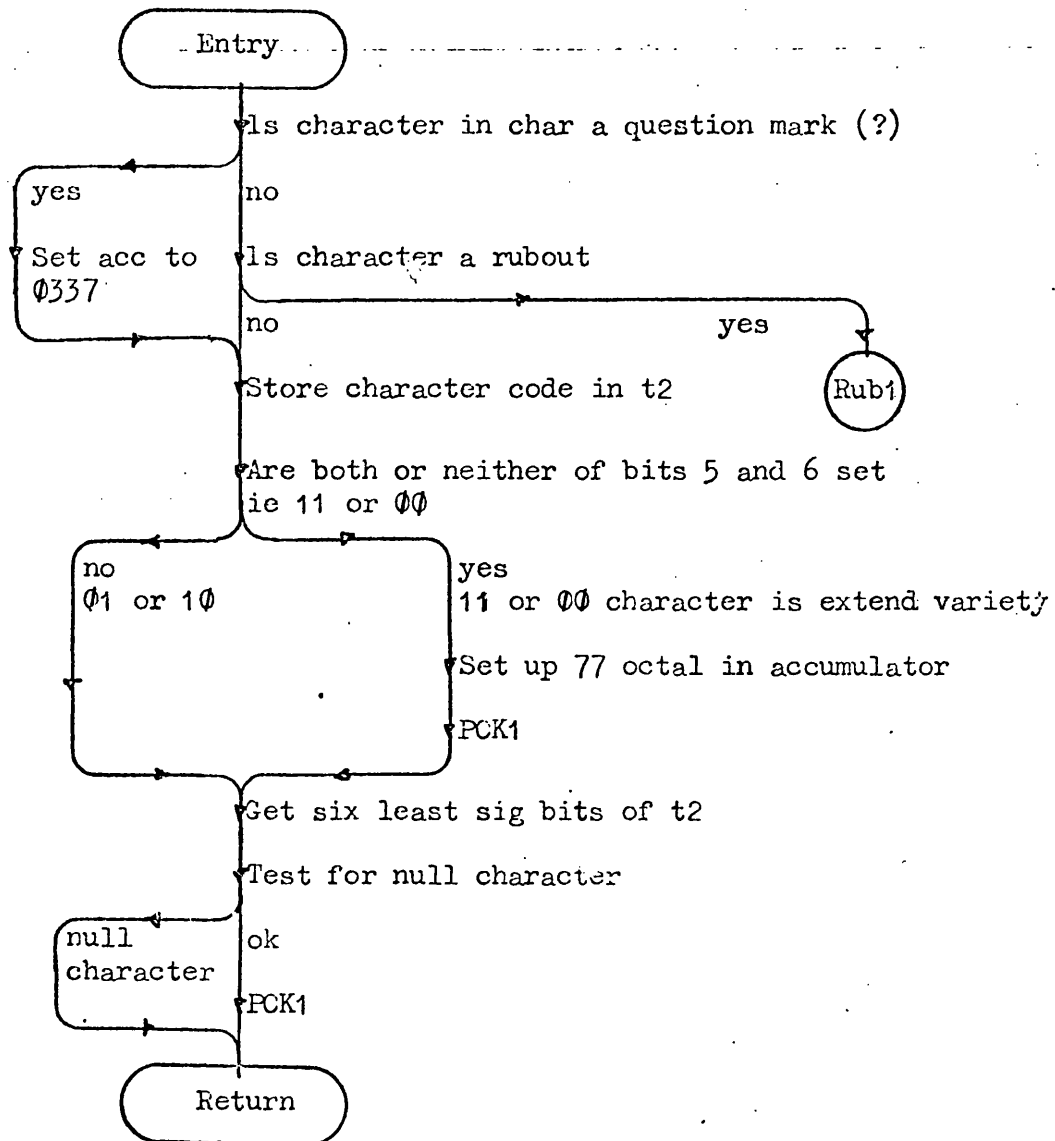
Chin

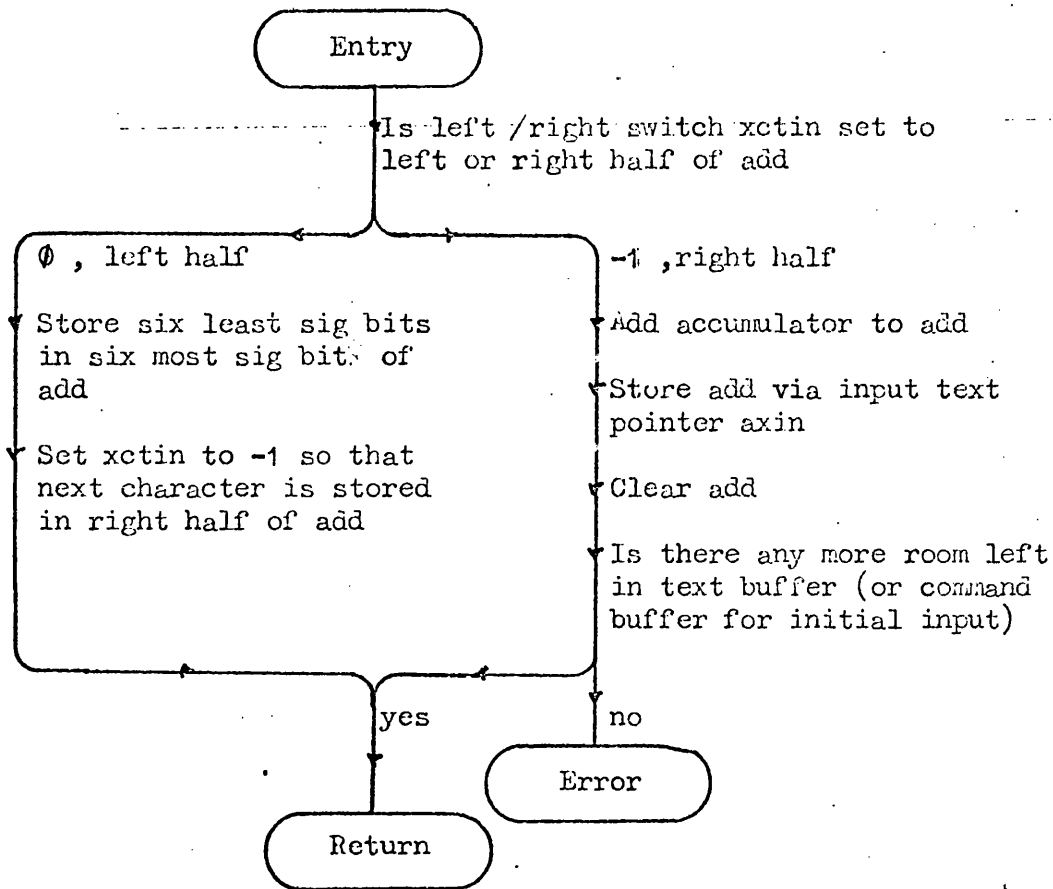
Called by READC



PACBUF . Strips and Packs ASCII Characters

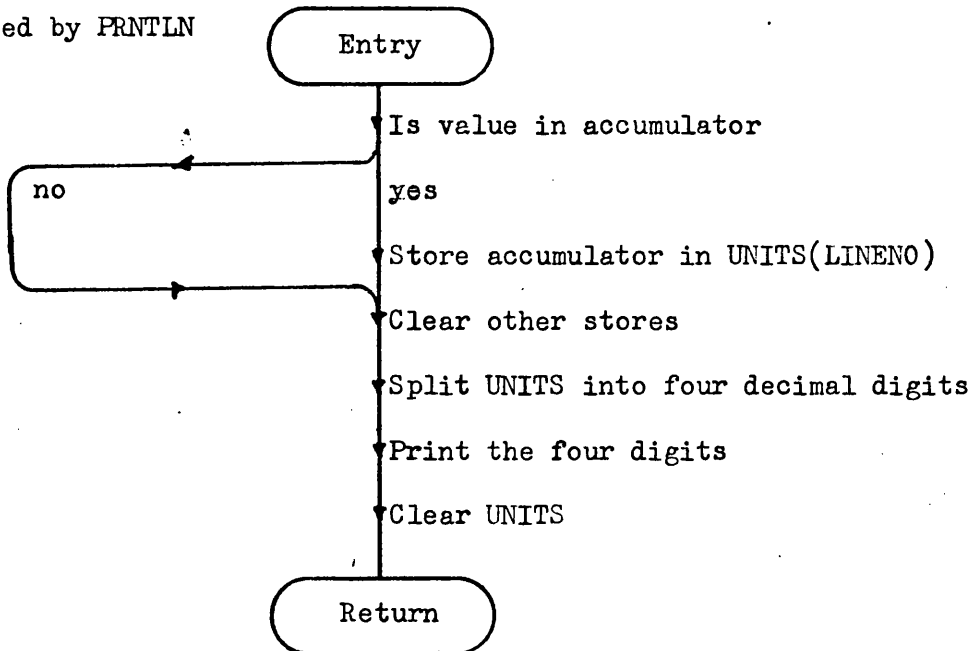
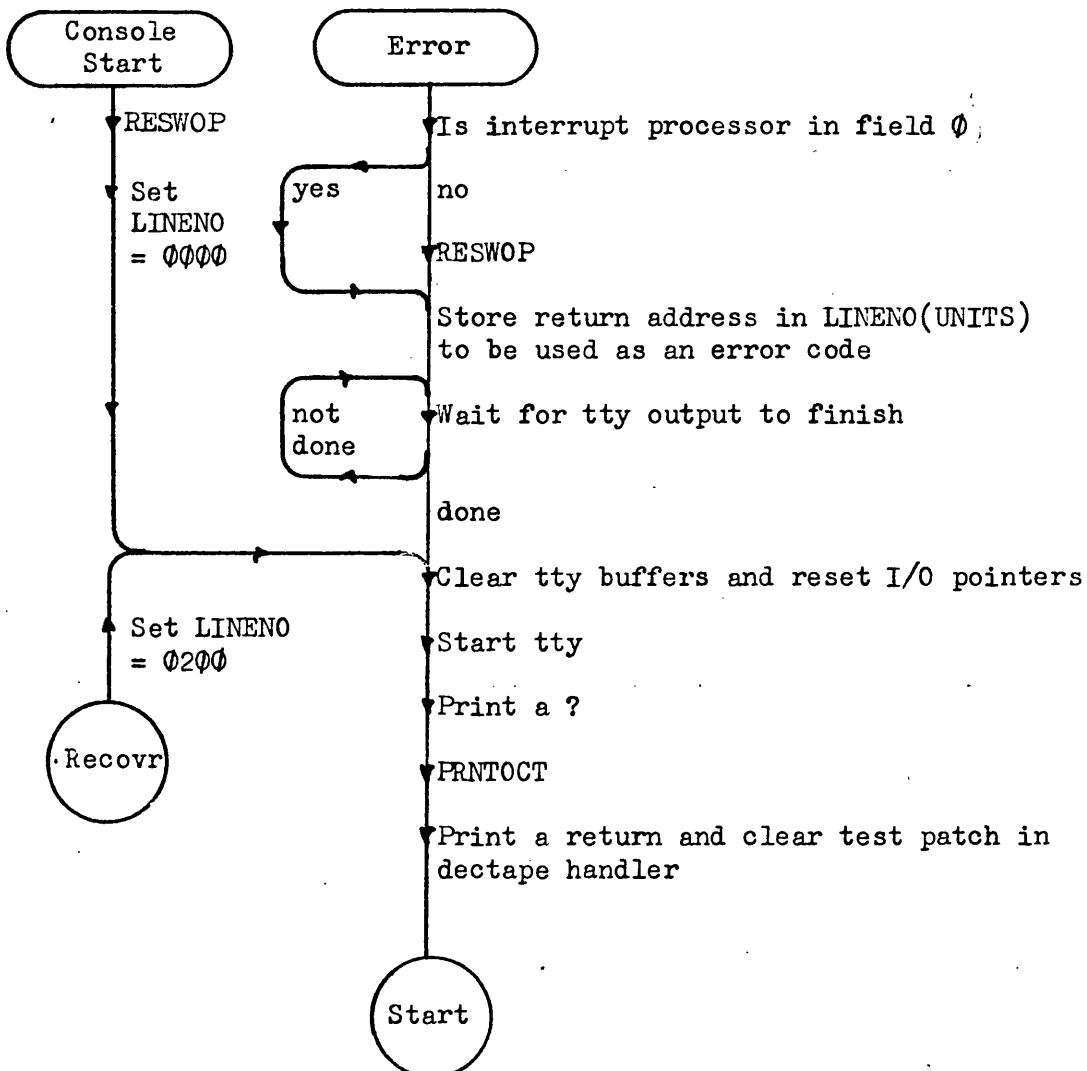
Called by PACKC

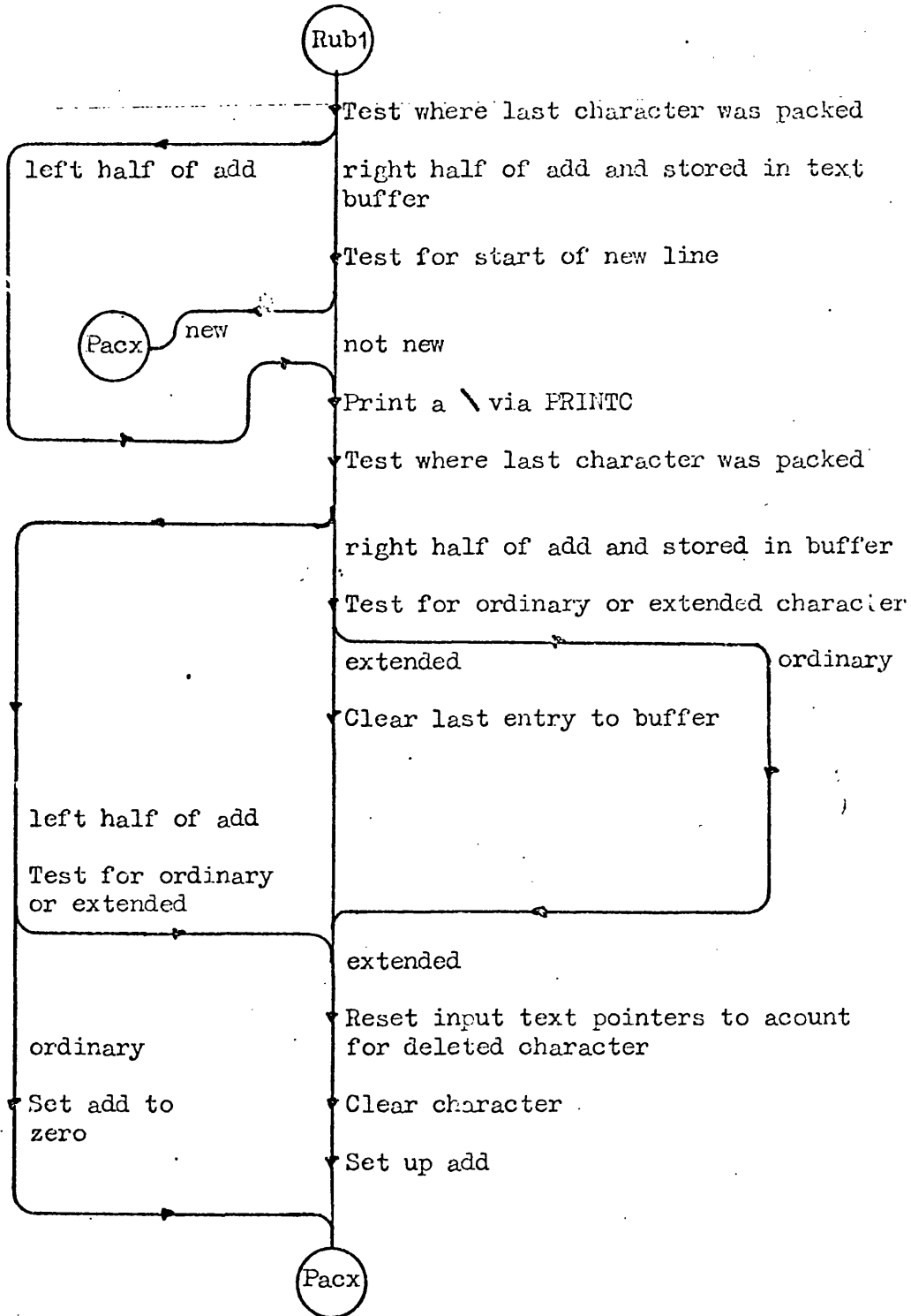


PCK1

Xprnt

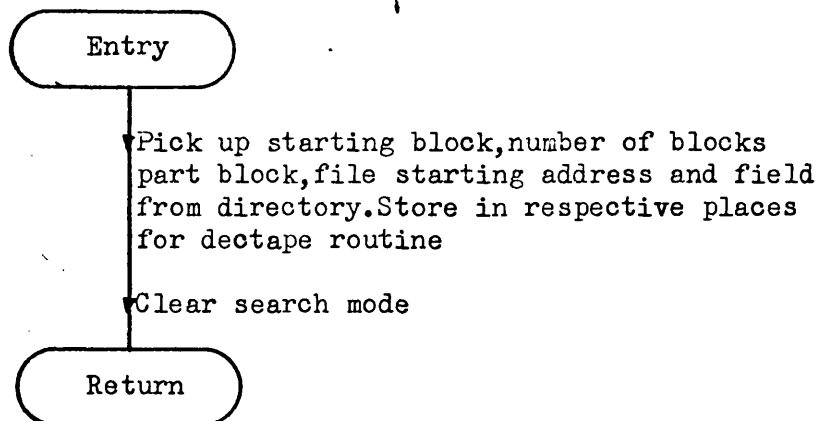
Called by PRNTLN

Console Restart and Error Recovery

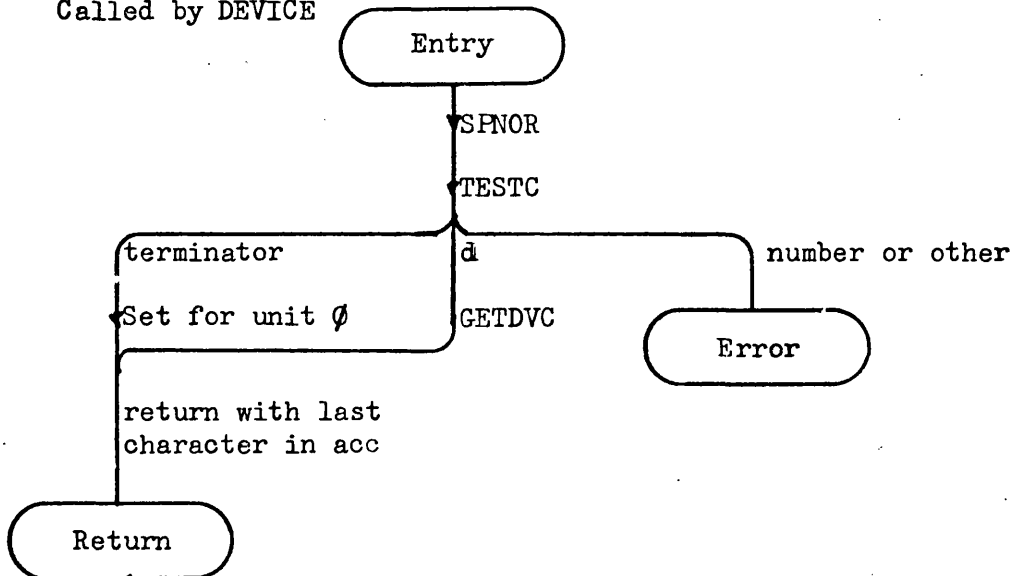
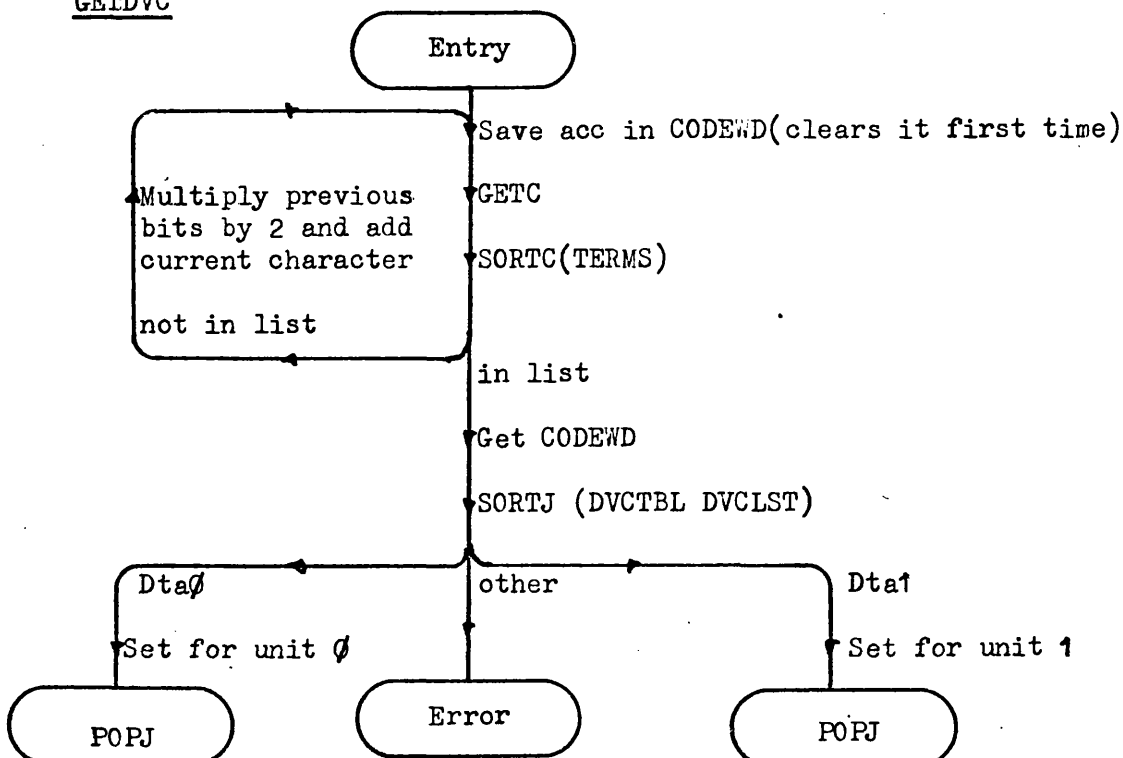
Character Removal Routine

Filestr

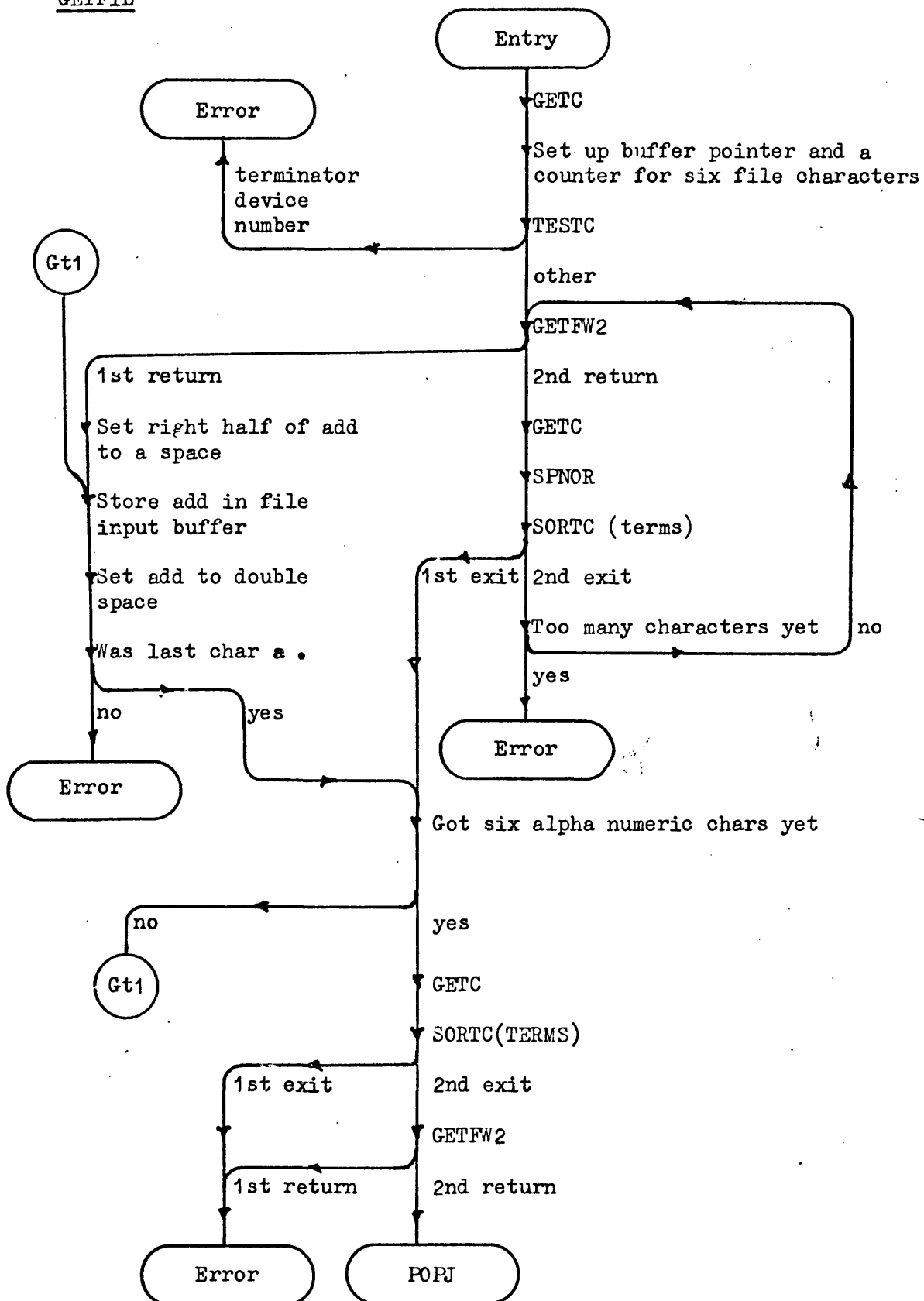
Called by FILSTR

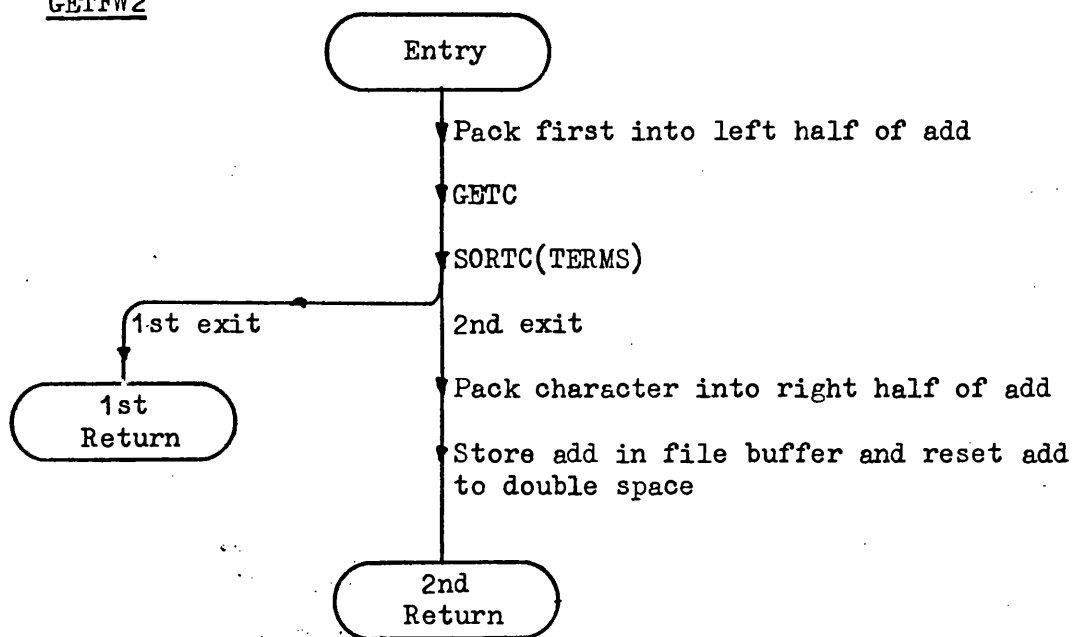
Dvce

Called by DEVICE

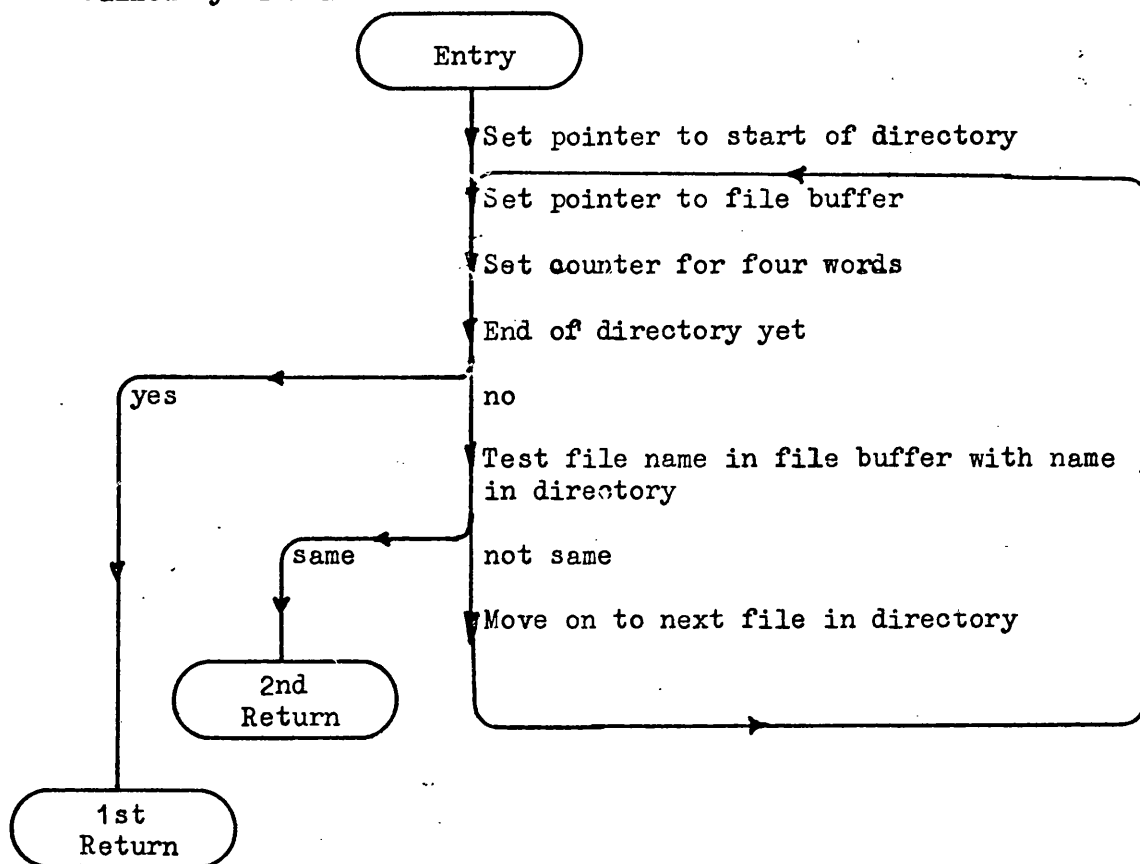
GETDVC

GETFIL

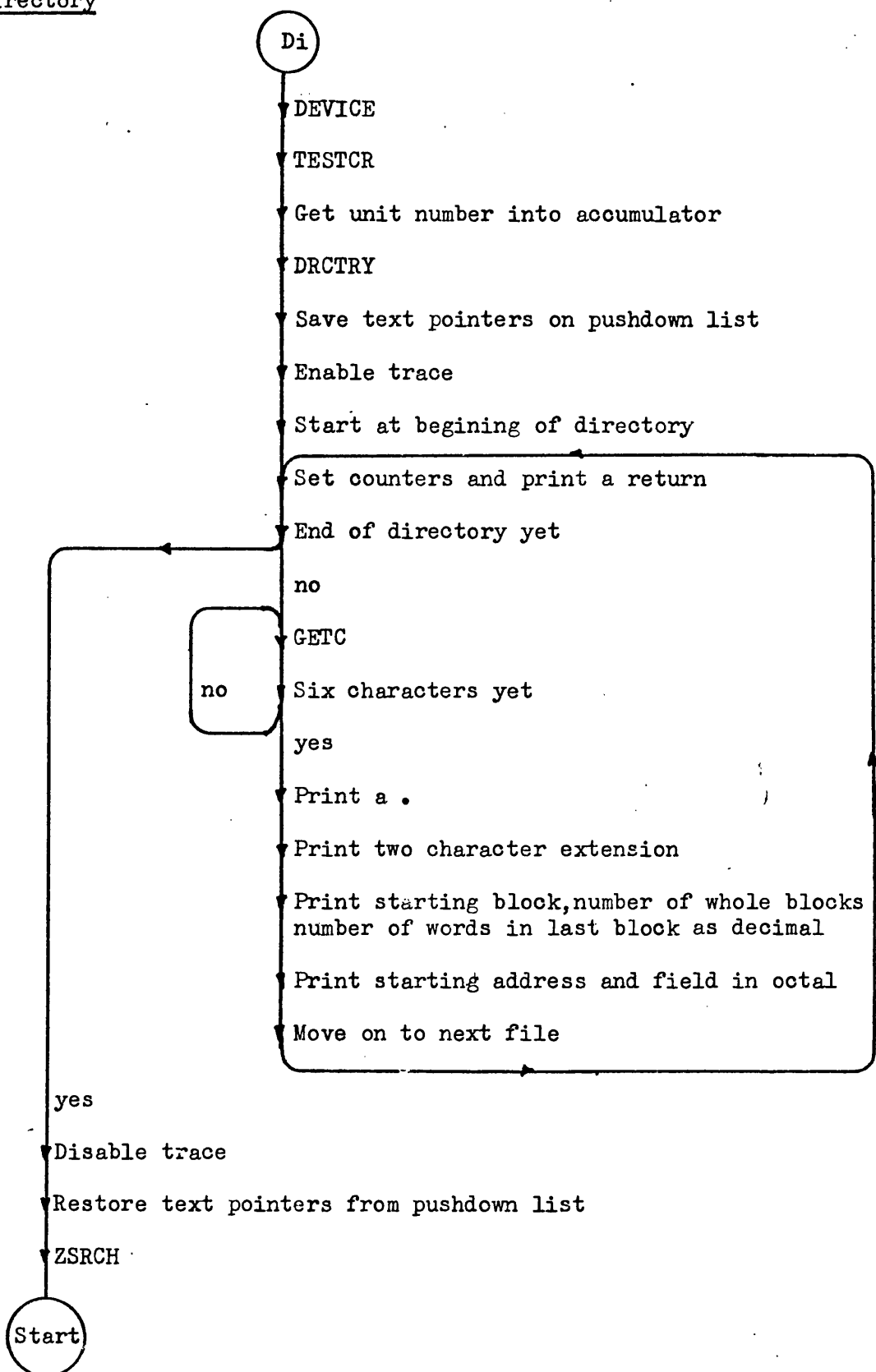


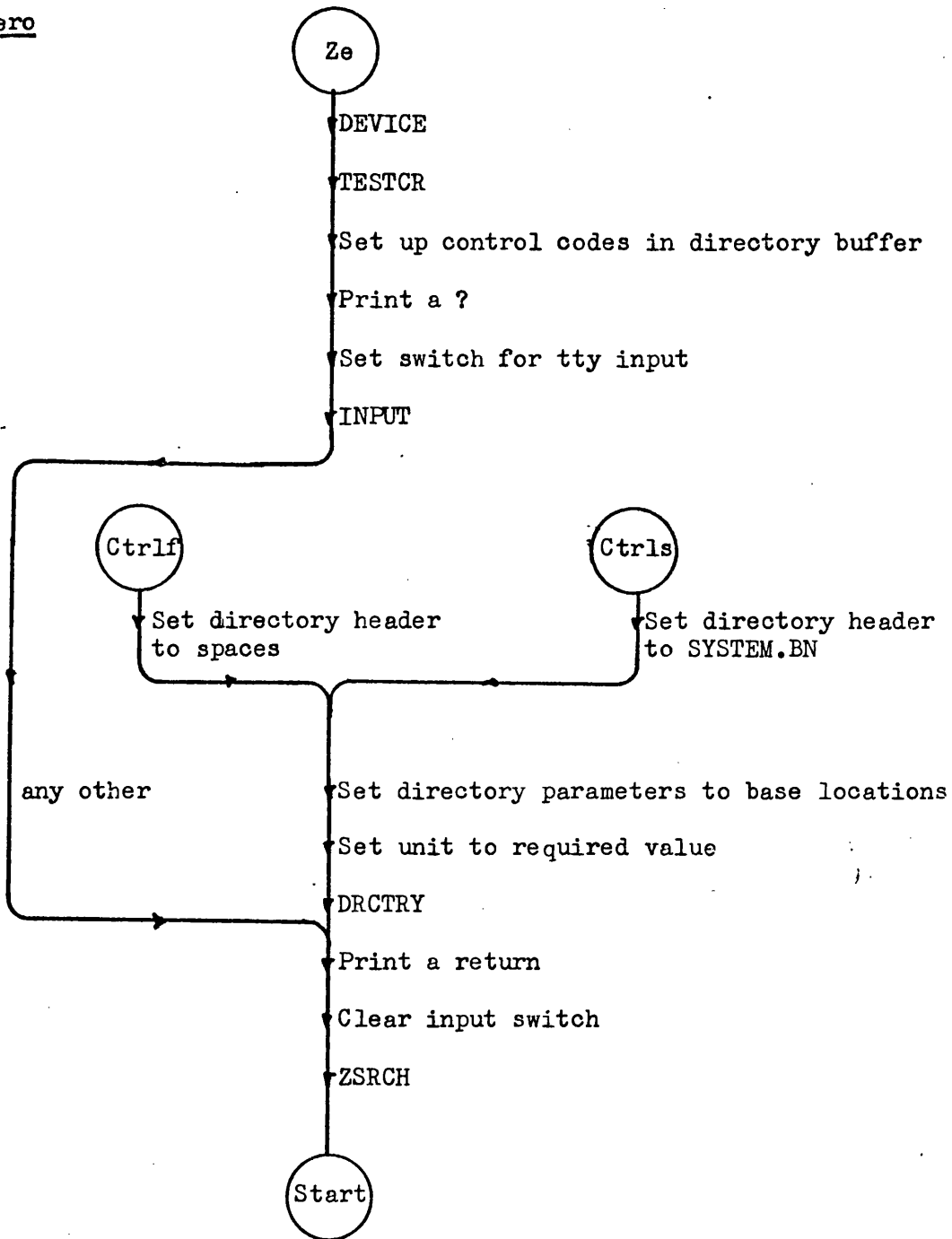
GETFW2Xsearch

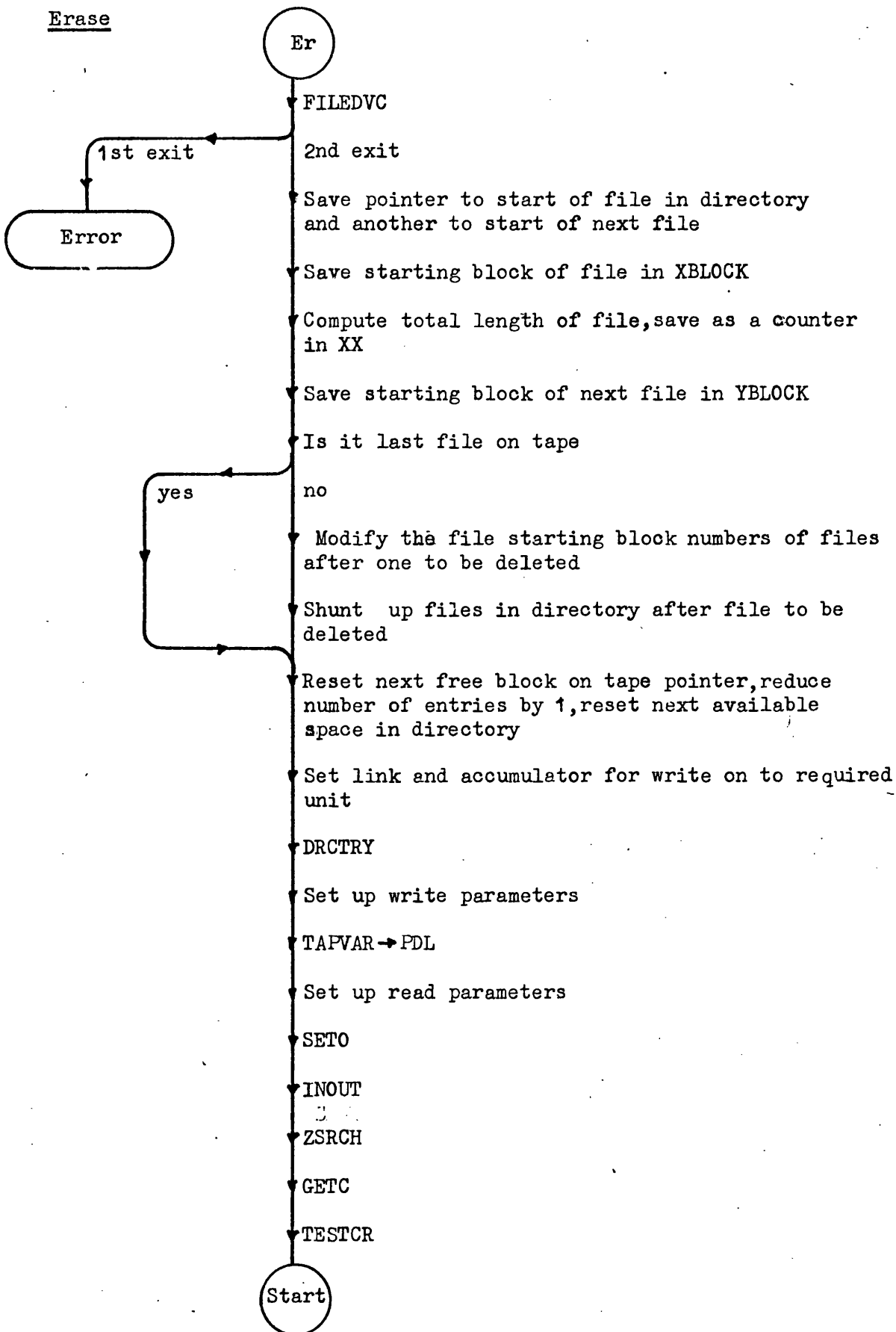
Called by FLSRCH

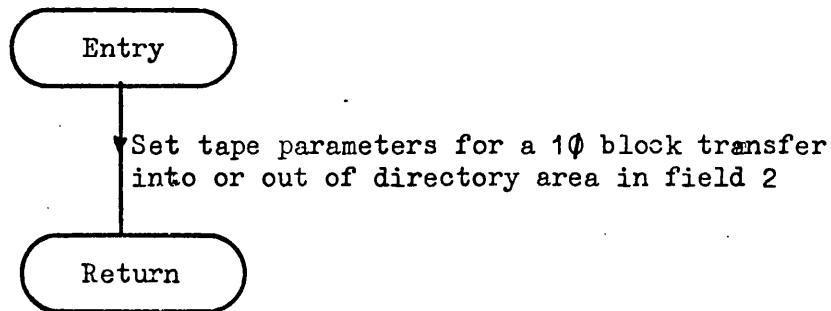
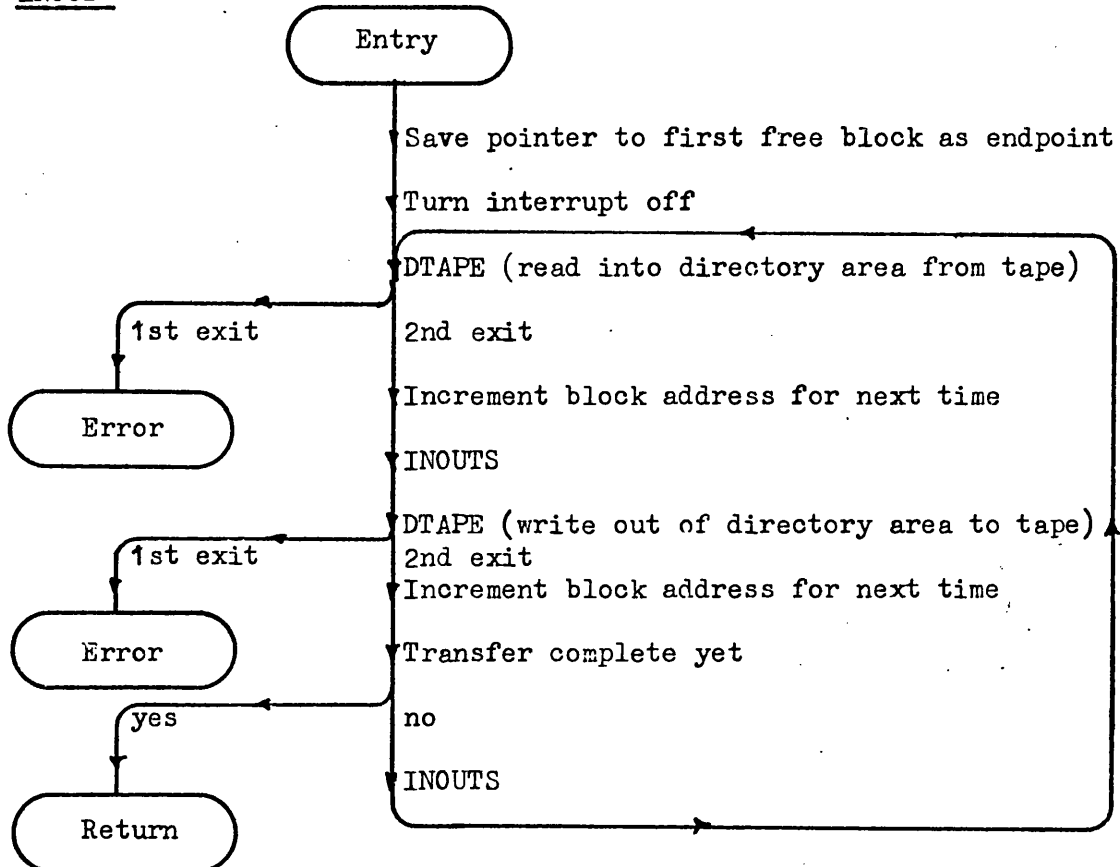
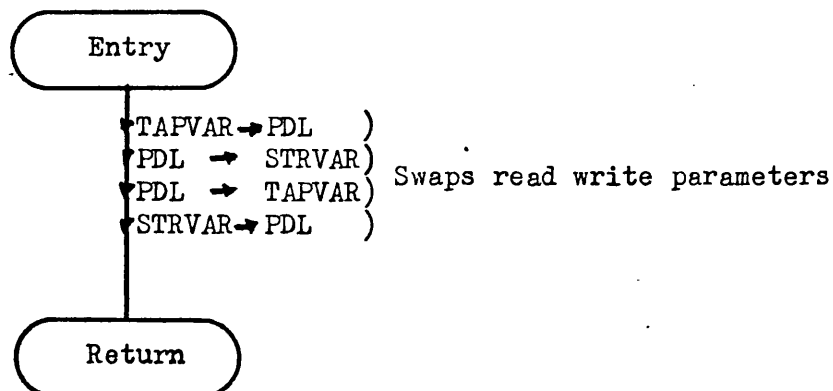


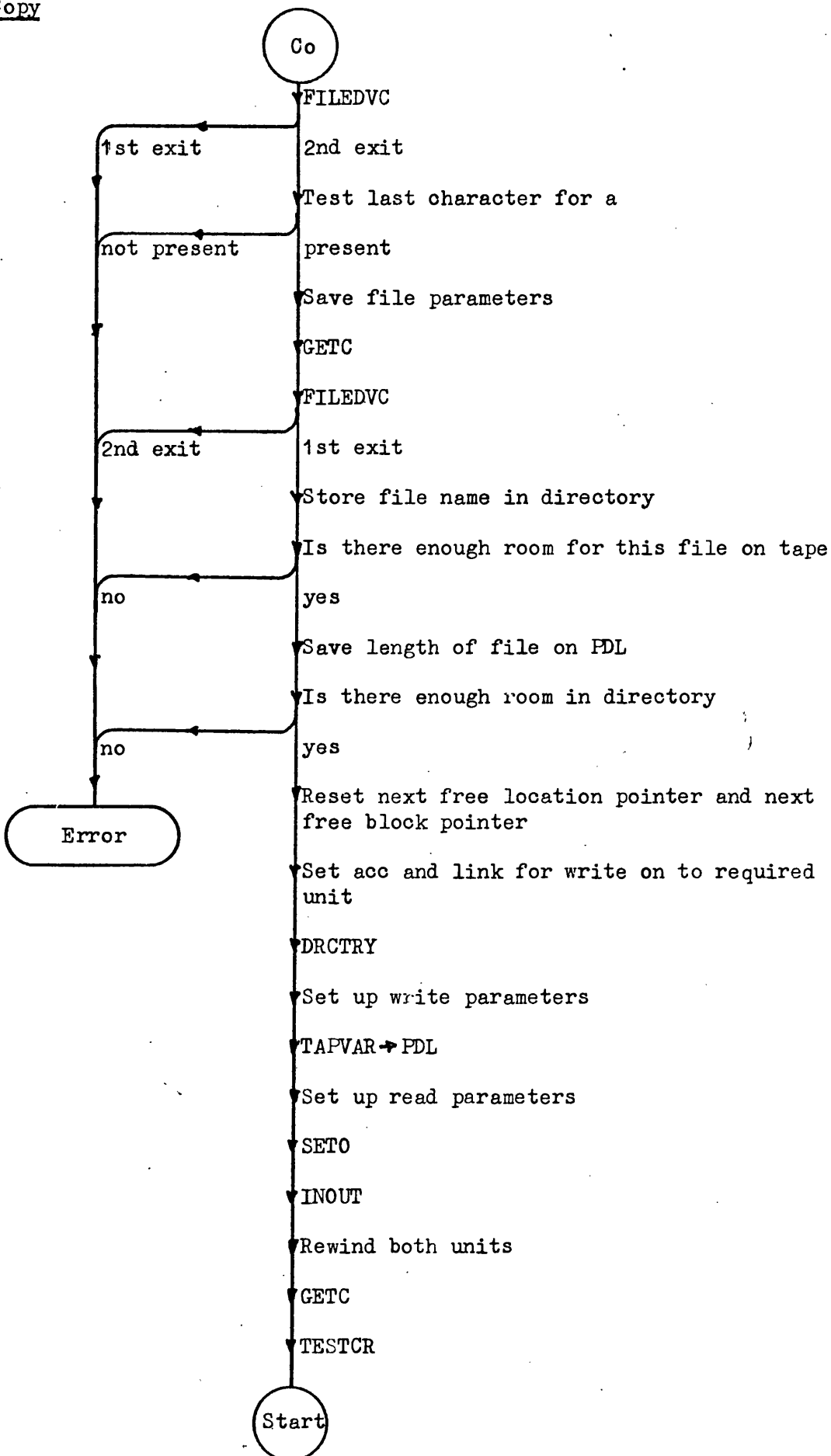
Directory



Zero

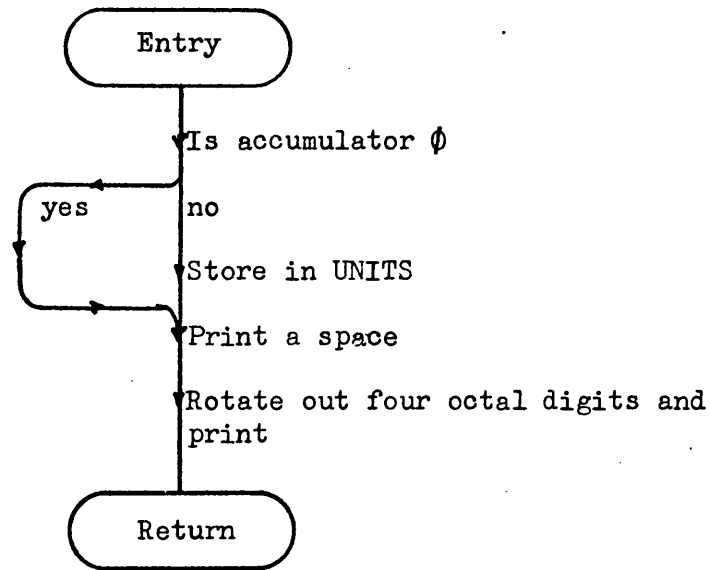
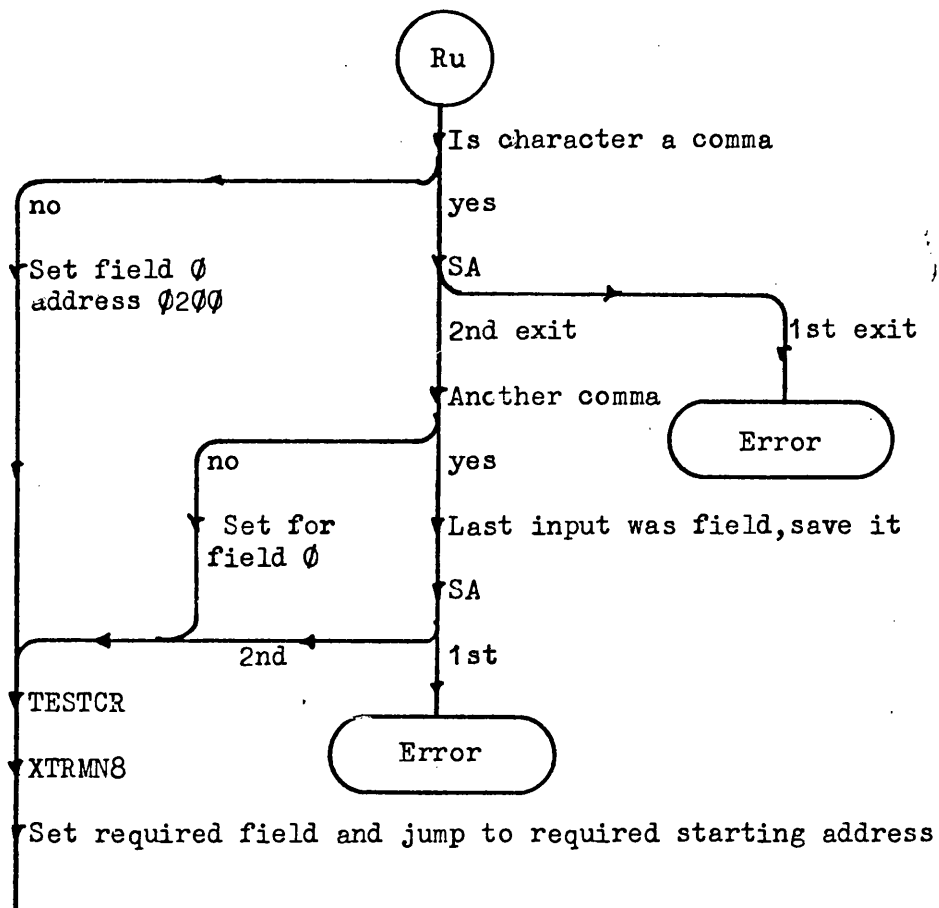
Erase

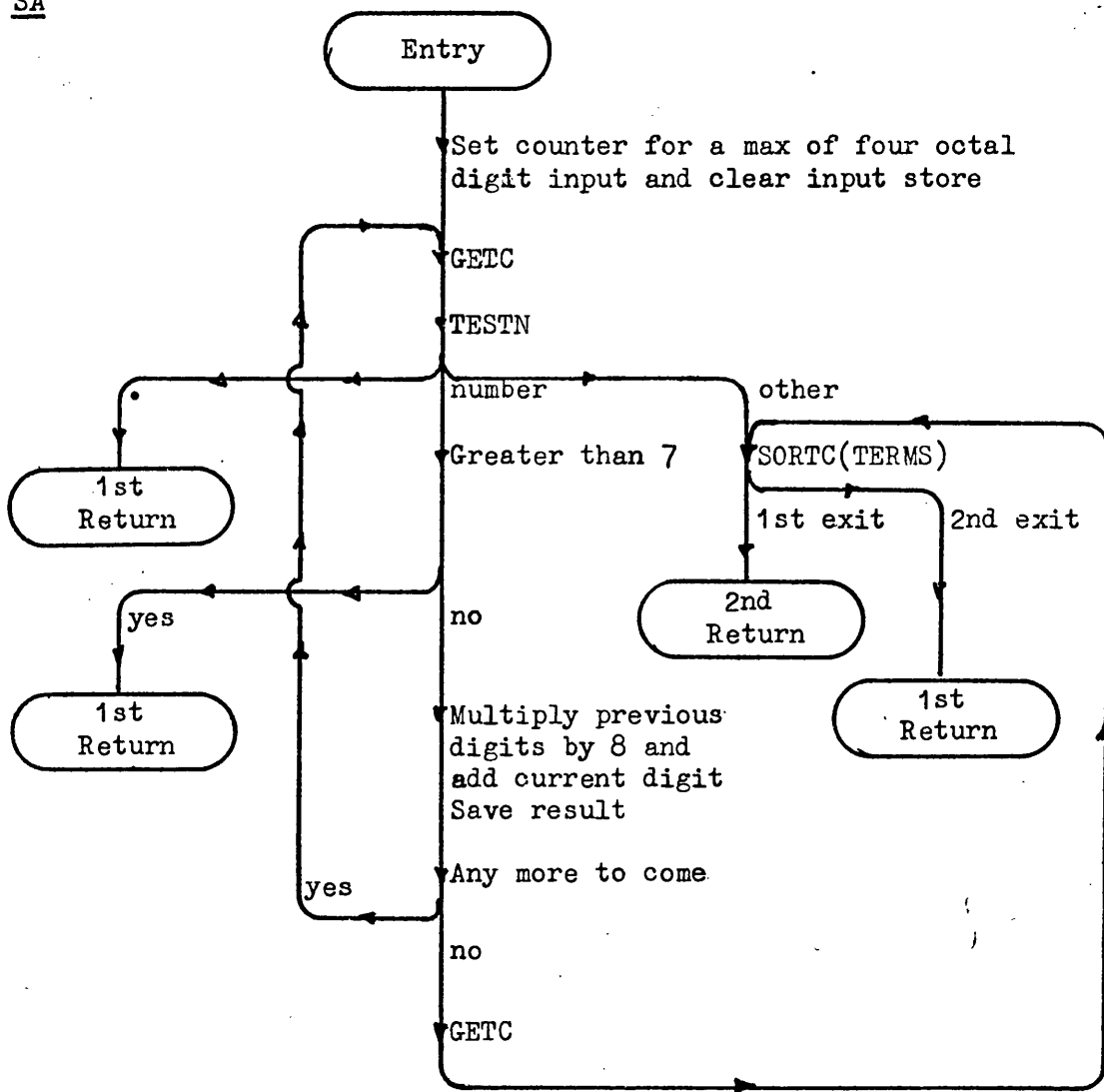
SET0INOUTINOUTS

Copy

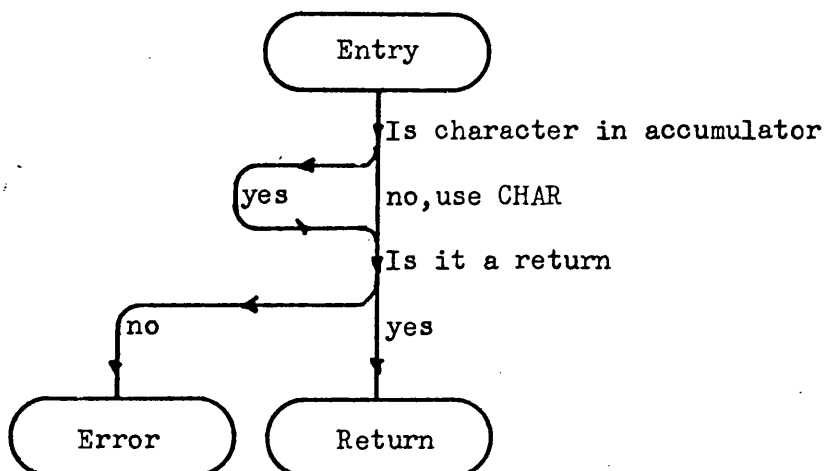
OCTPRINT

Called by PRNTOCT

Run

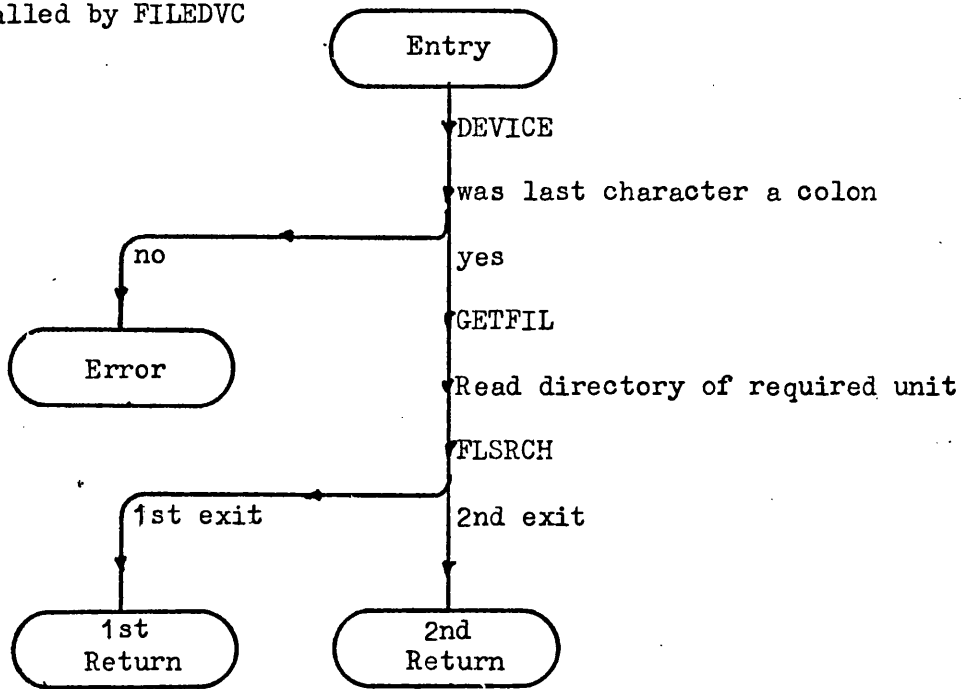
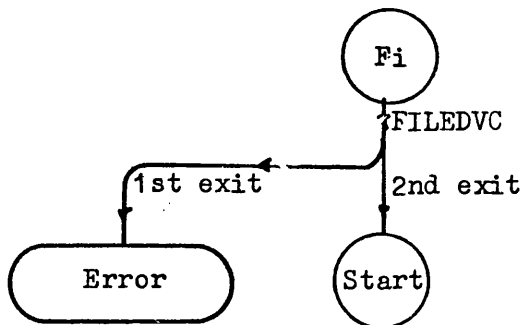
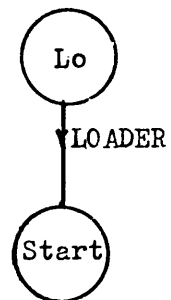
SATCARTN

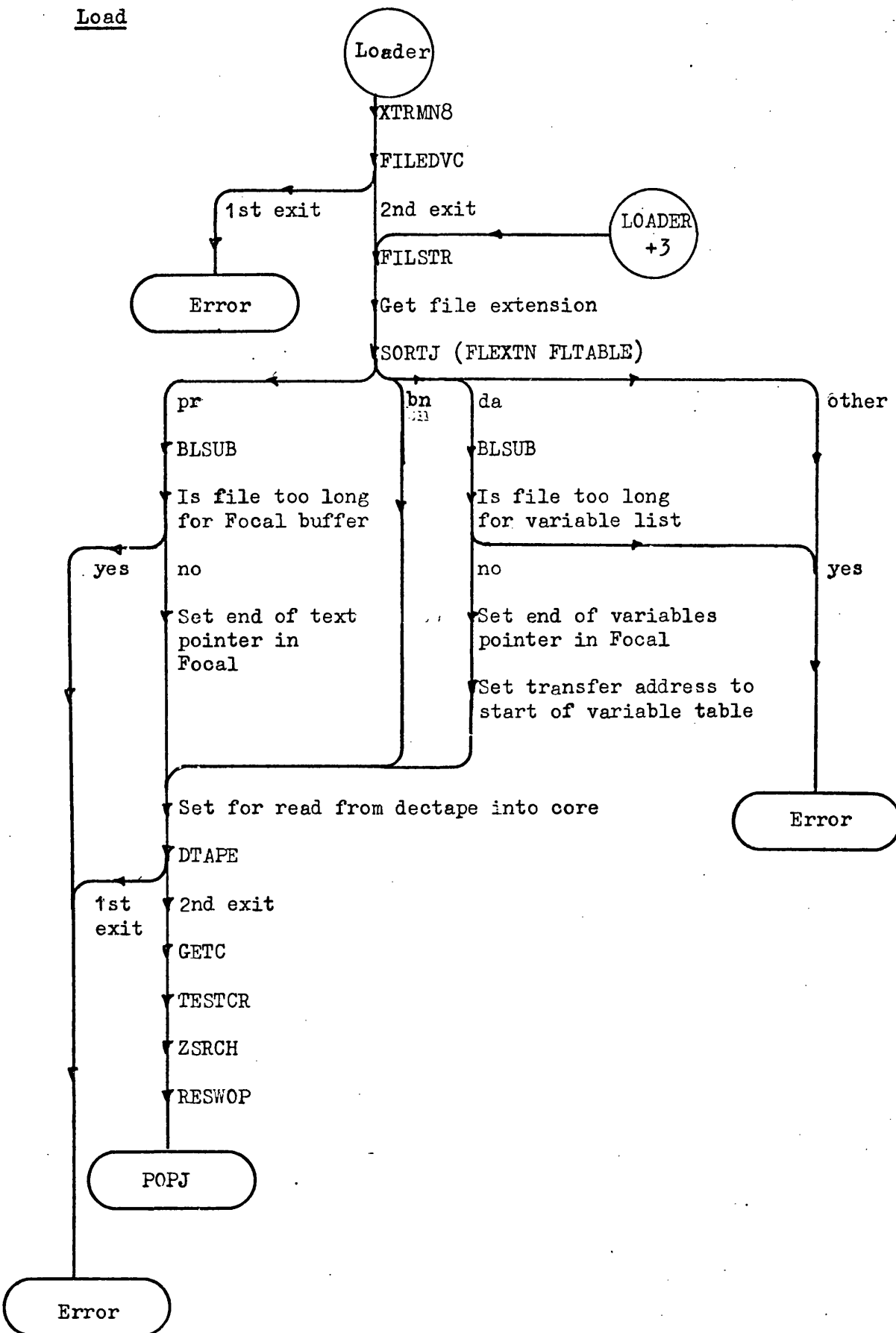
Called by TESTCR

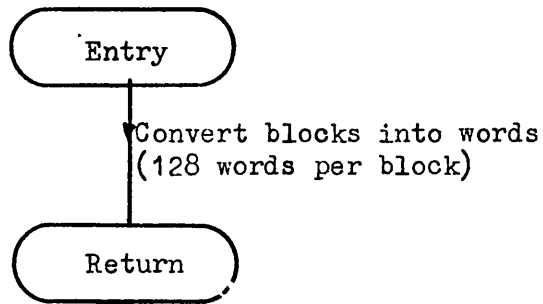


DVCFLE

Called by FILEDVC

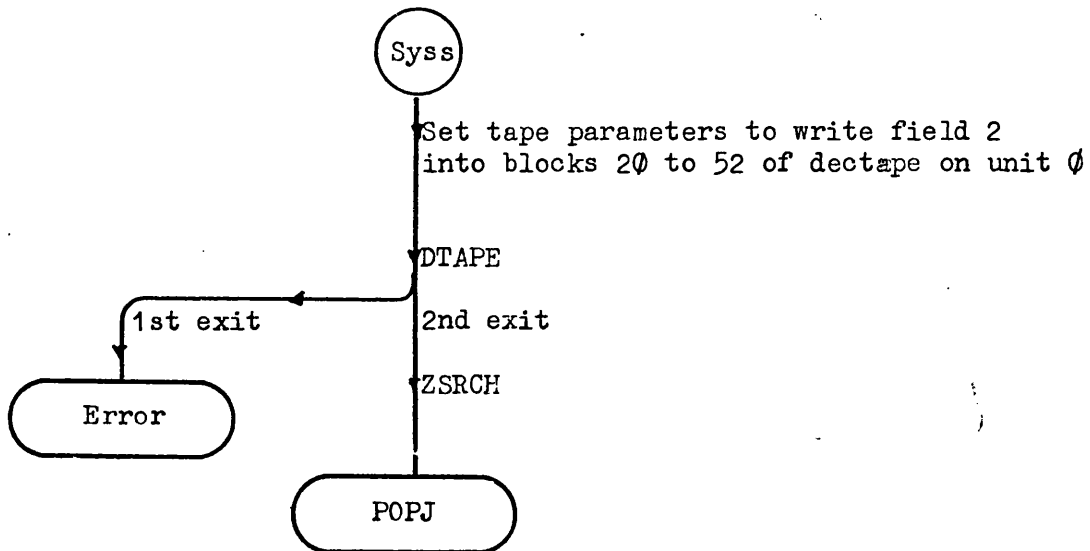
FindLoad

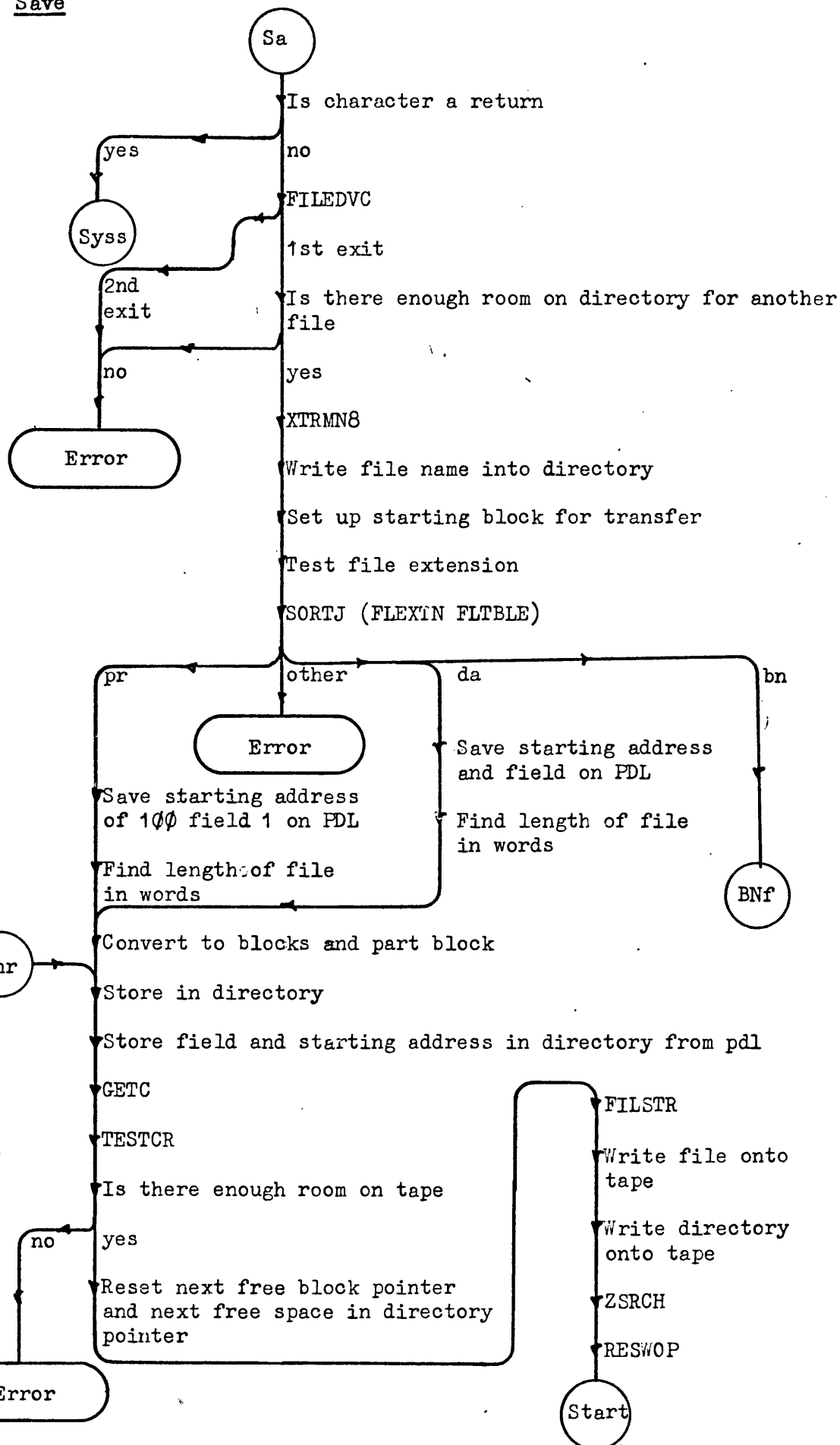
Load

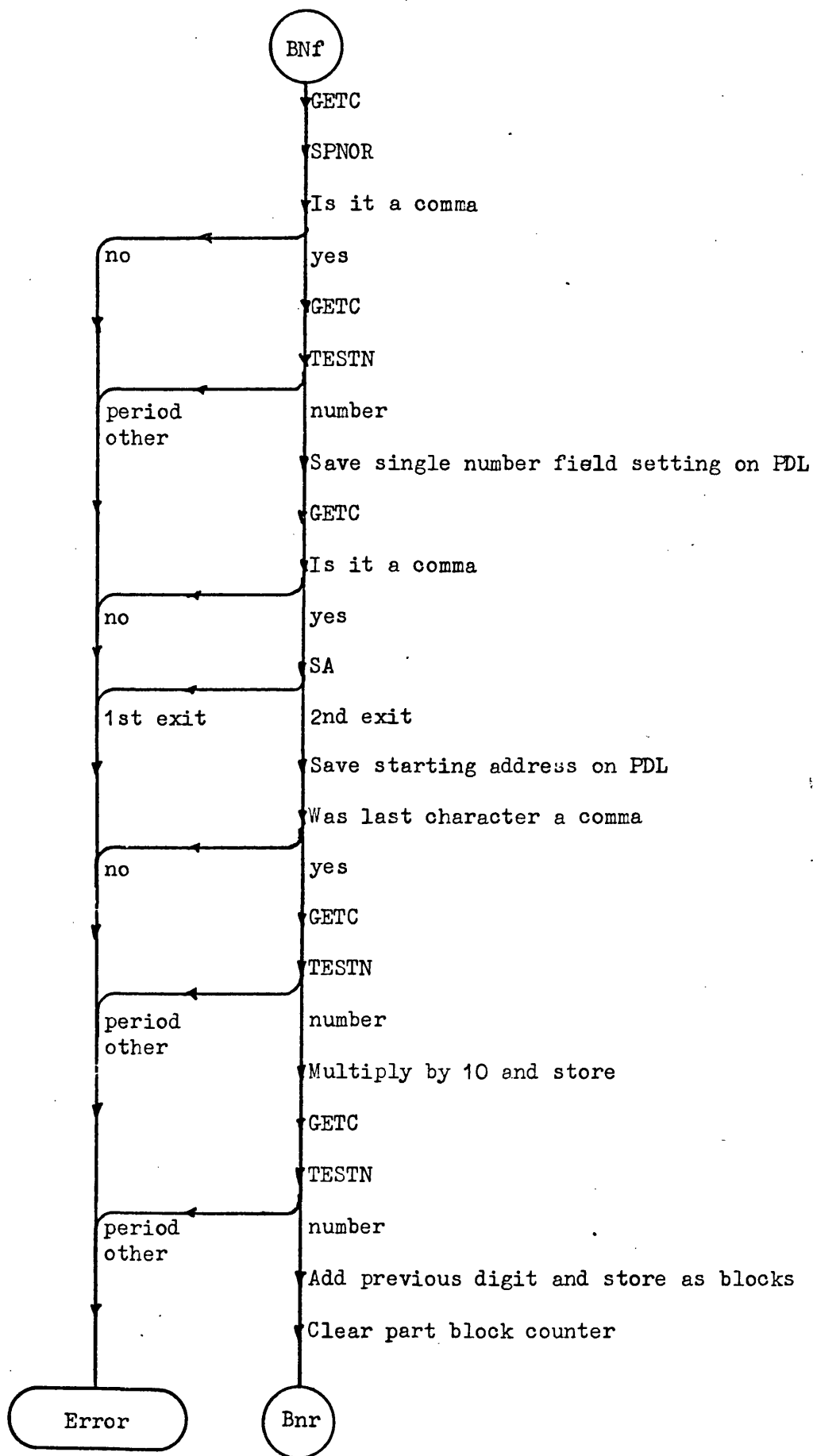
BLSUBSYSS

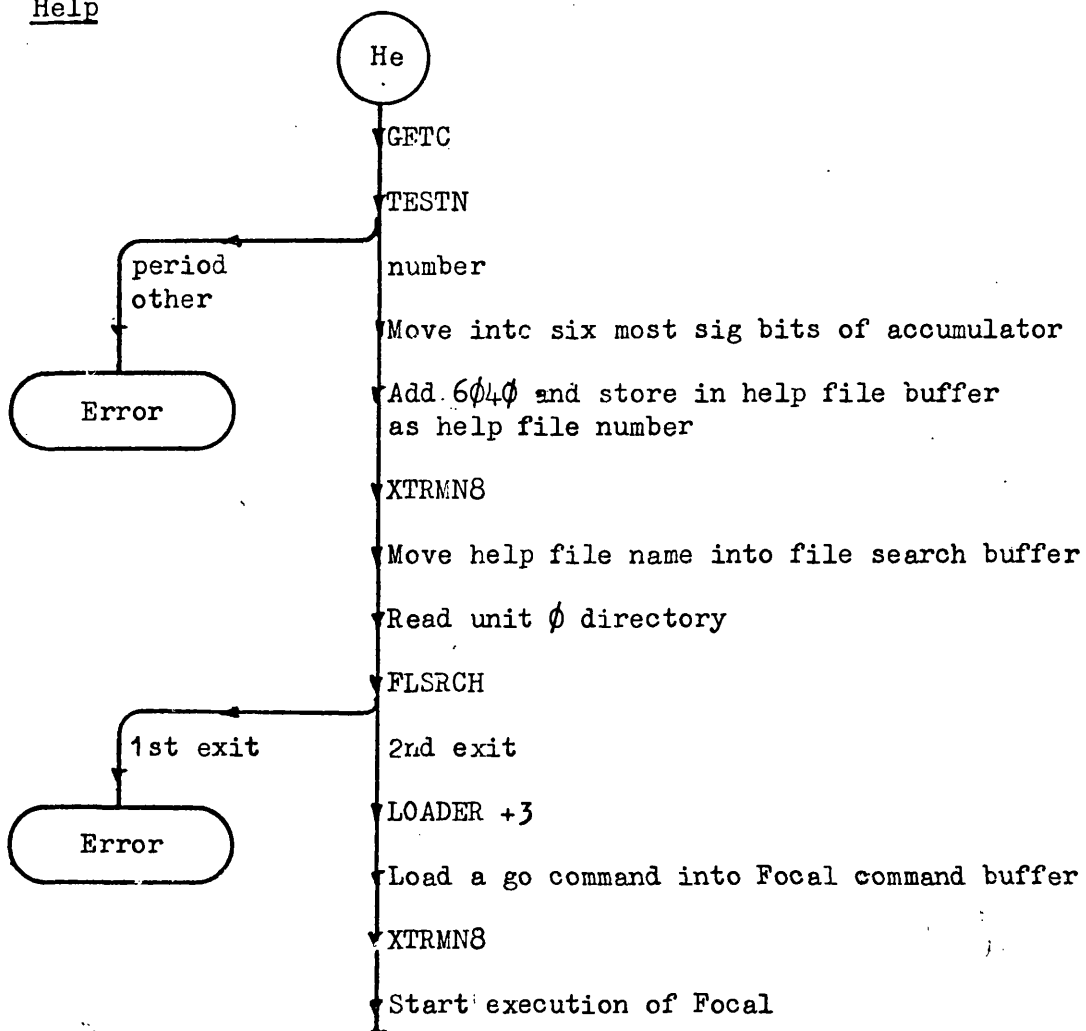
Section for saving system area on dectape

Extension of the save command



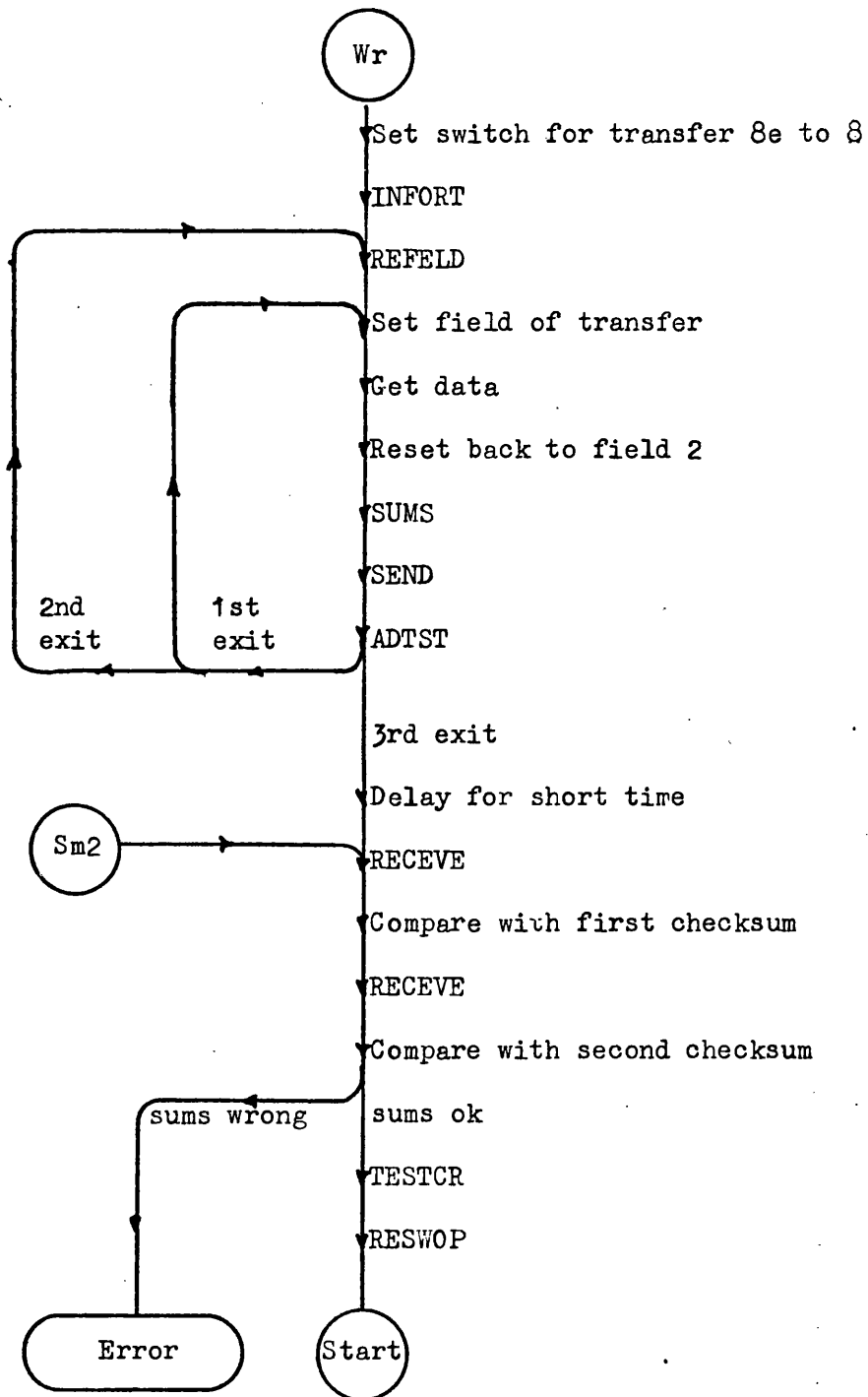
Save



Help

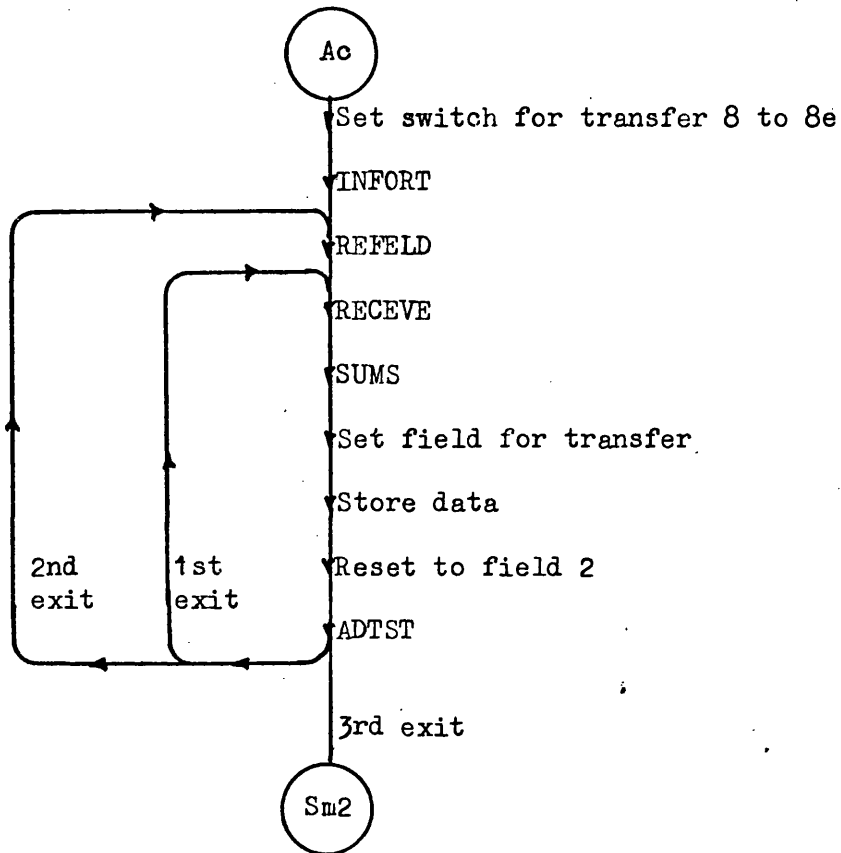
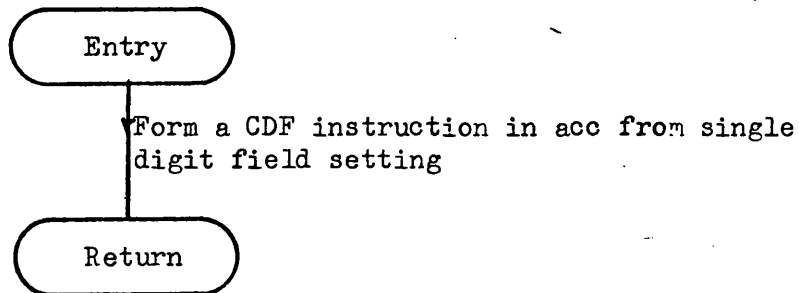
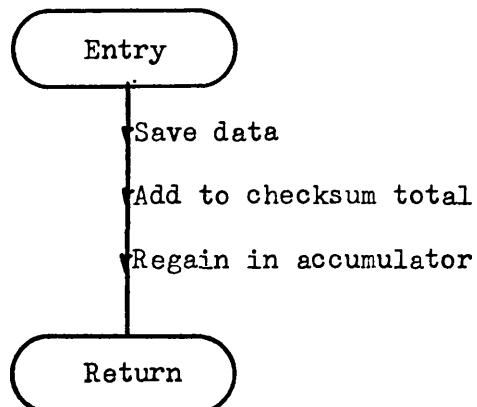
Write

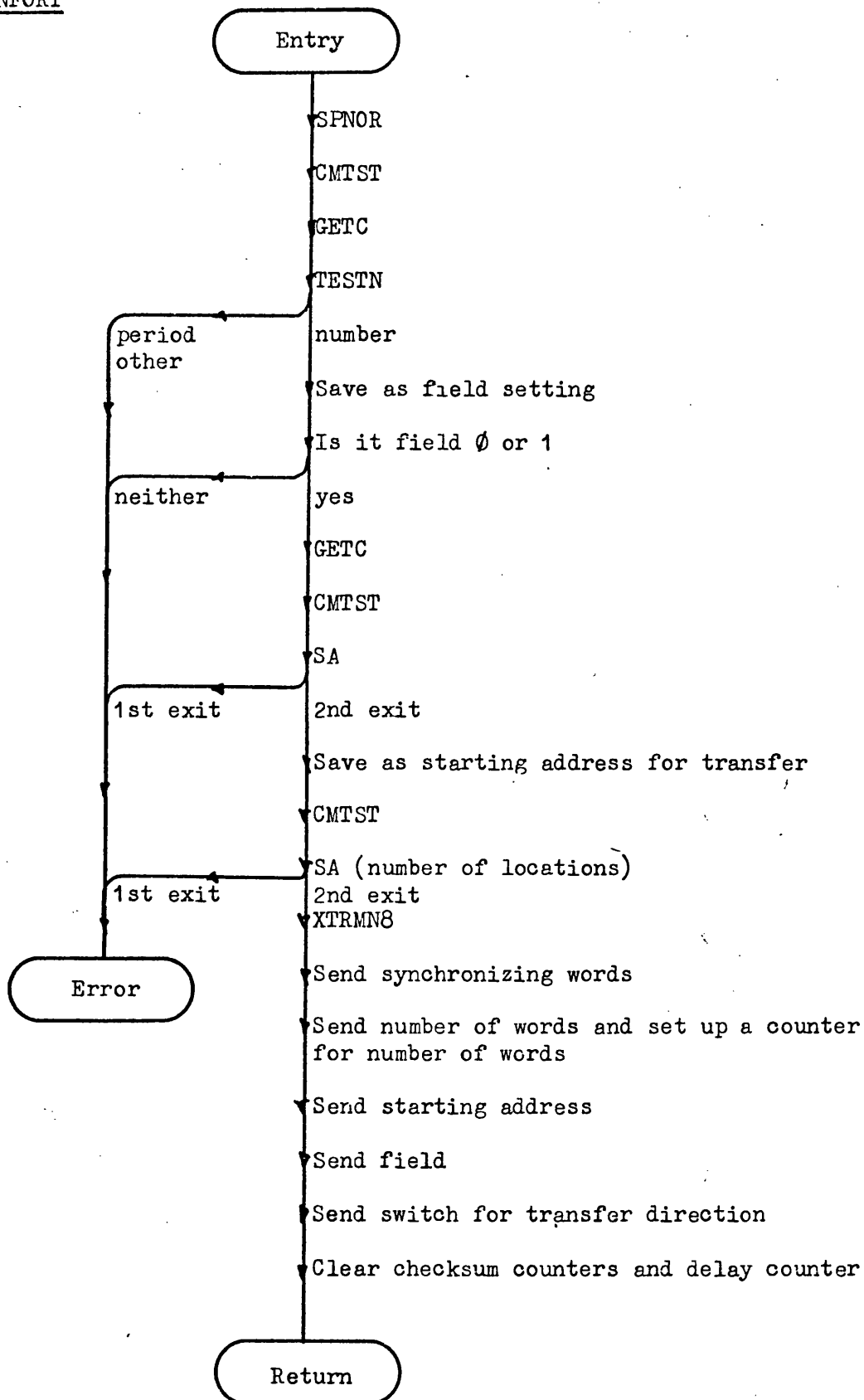
For use with Inter Processor Buffer

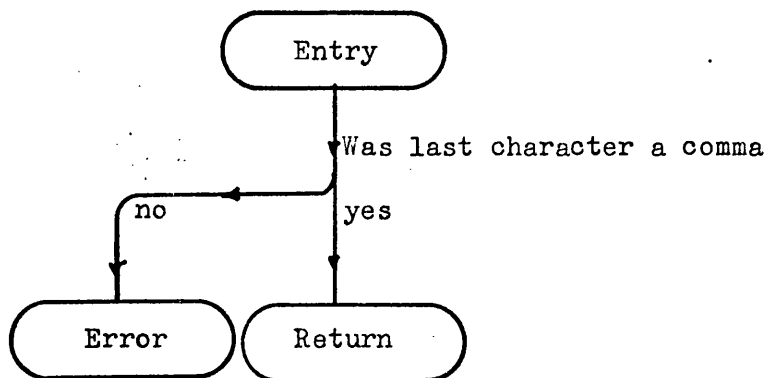
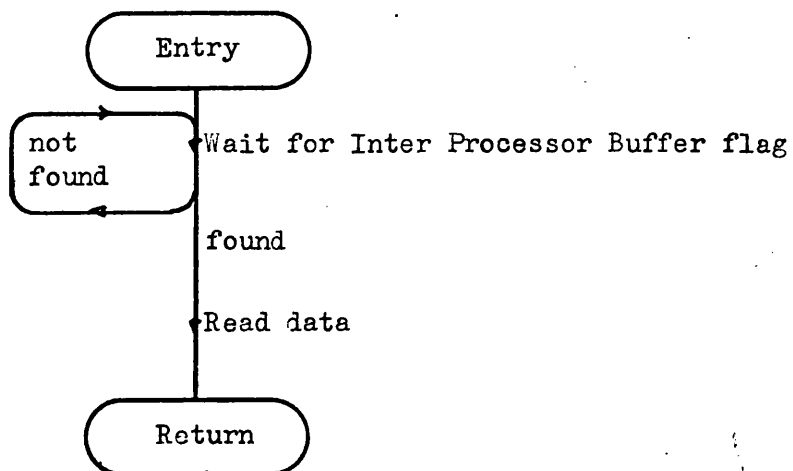
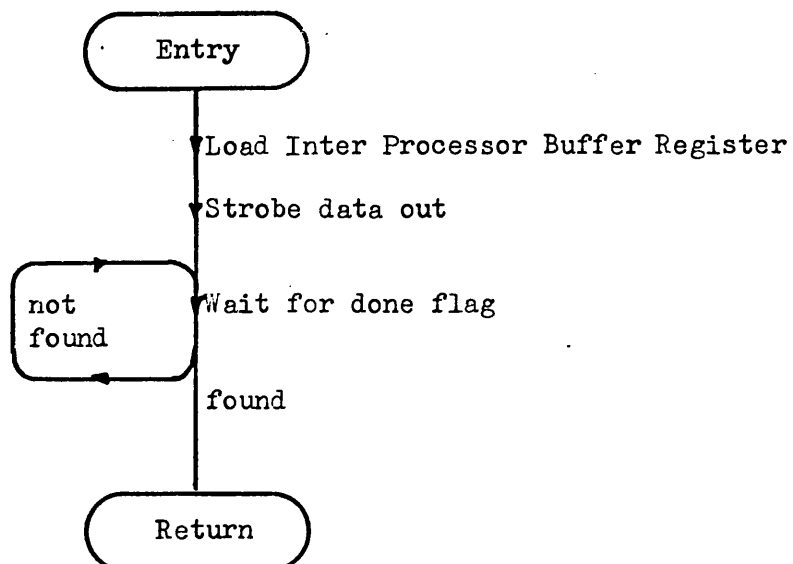


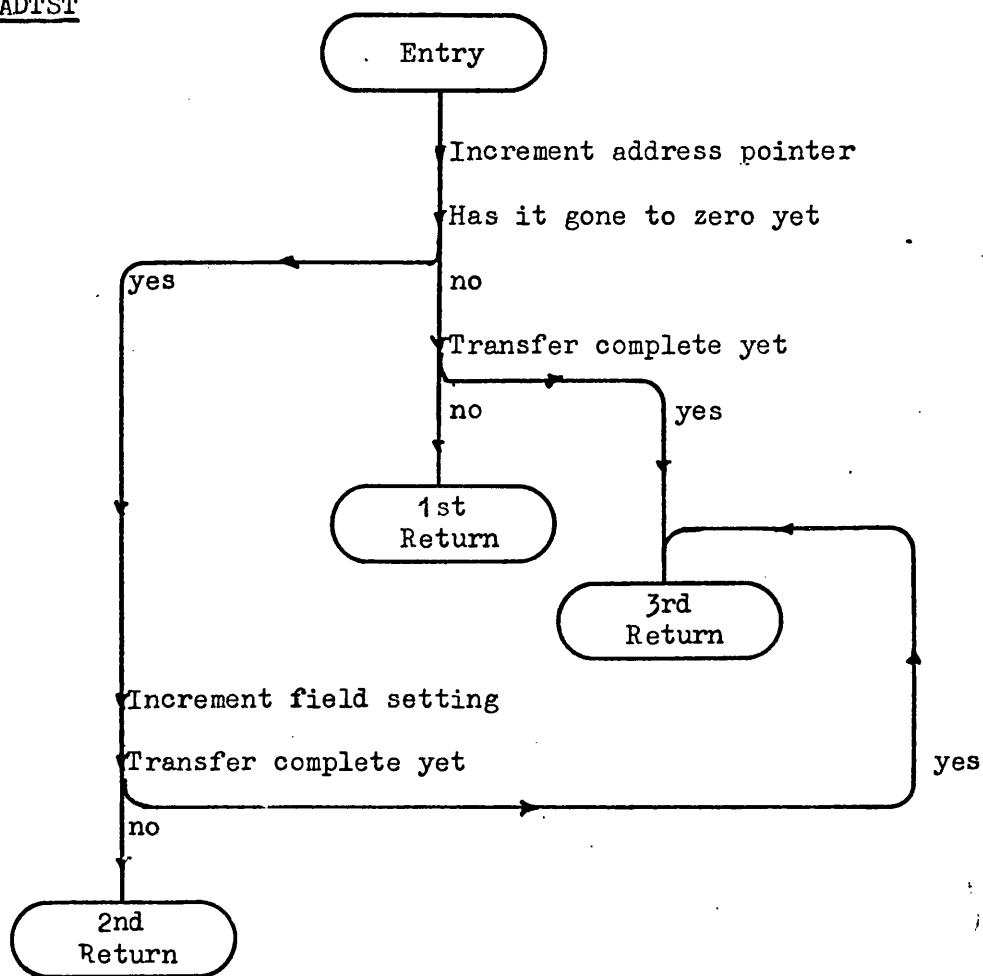
Access

For use with Inter Processor Buffer .

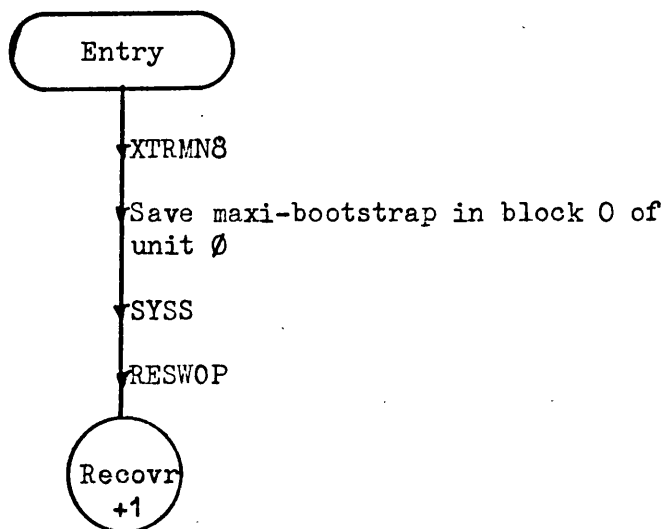
REFELDSUMS

INFORT

CMTSTRECEVESEND

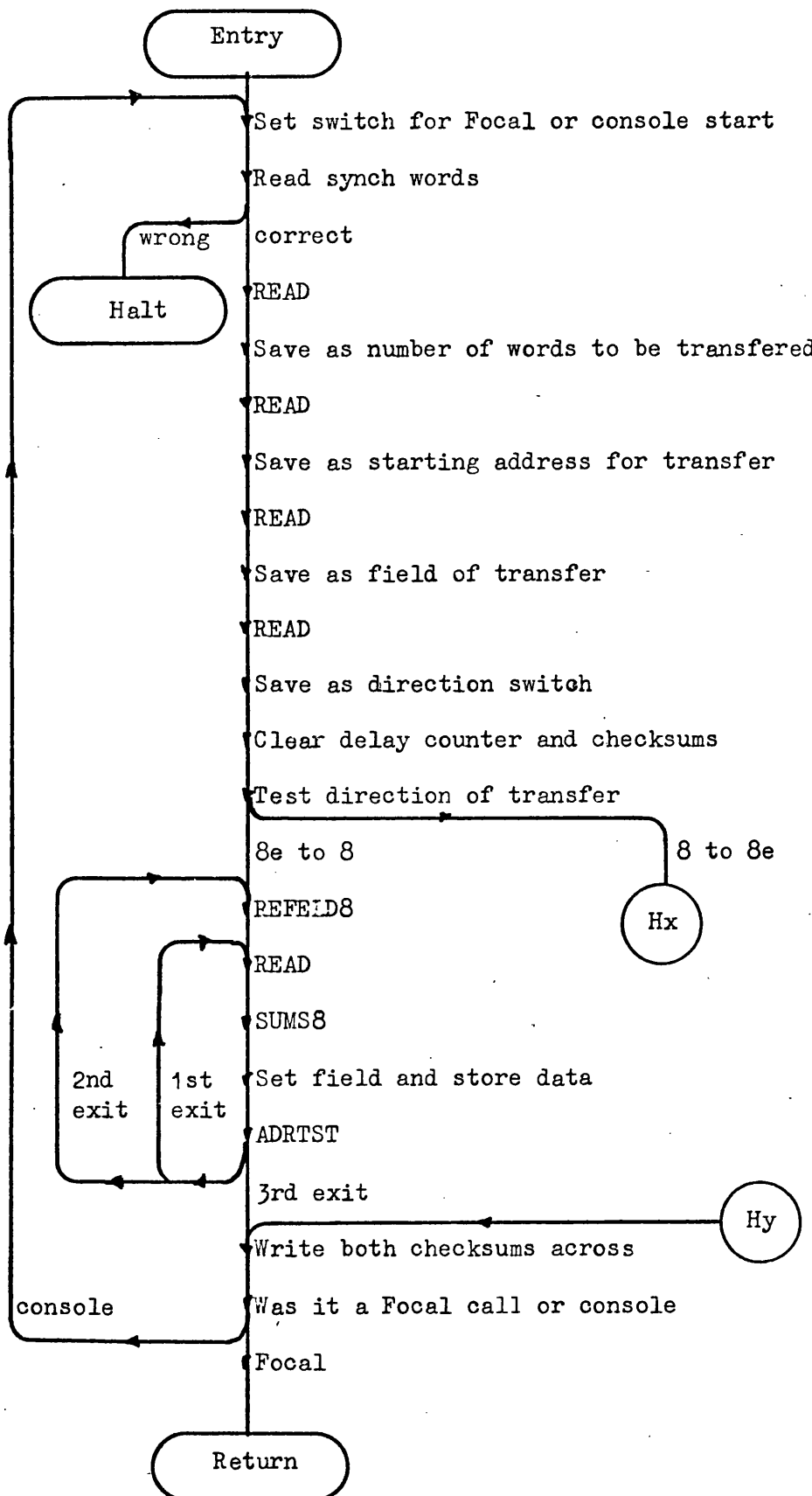
ADTSTBEGINN

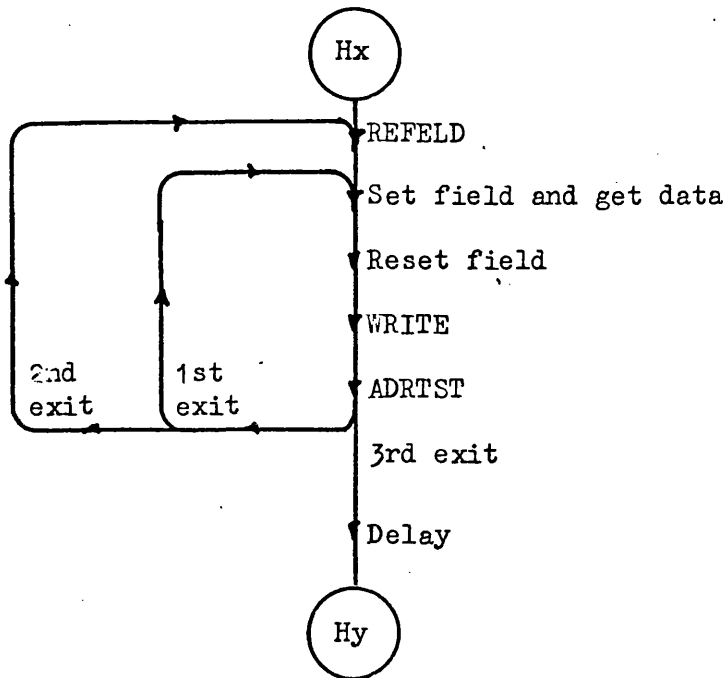
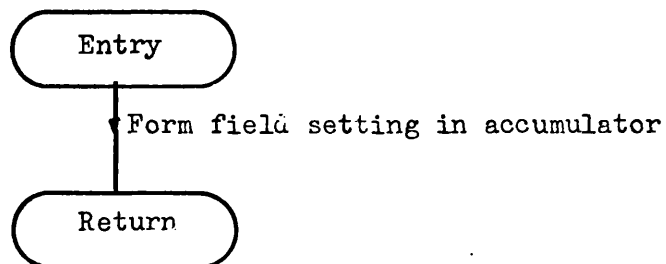
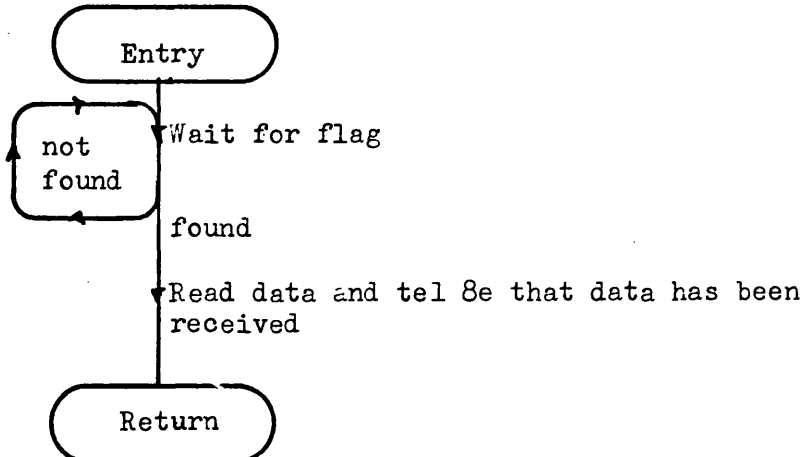
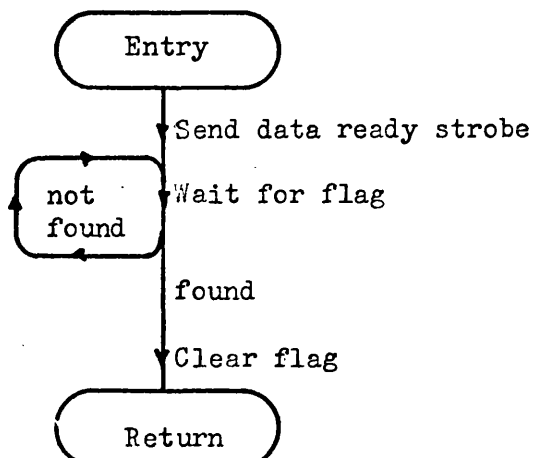
Used when building the system from paper tape

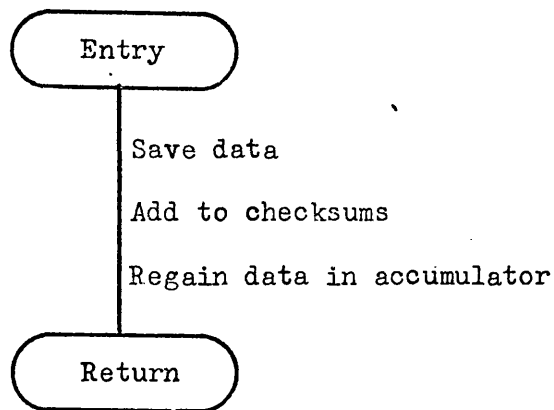


PDP-8 Inter Processor Buffer Handler

Takes the place of the loaders in field 1



REFELD8READWRITE

SUMS8ADRTST